МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| Разработка приложения для организации взаимодействия объектов при заданных критериях |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |

РАБОТУ ВЫПОЛНИЛА

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР. | 4932 |  |  |  | Е.А. Цыганкова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

Оглавление

[1. Введение 3](#_Toc89630284)

[2. Задание на курсовое проектирование 3](#_Toc89630285)

[3. Постановка задачи 3](#_Toc89630286)

[3.1. Анализ предметной области 3](#_Toc89630287)

[3.2. Формулировка технического задания 3](#_Toc89630288)

[4. Проектирование классов 7](#_Toc89630289)

[4.1. Разработанные классы 7](#_Toc89630290)

[4.2. Диаграмма классов 7](#_Toc89630291)

[5. Разработка приложения 9](#_Toc89630292)

[5.1. Разработка интерфейса приложения 9](#_Toc89630293)

[5.2. Реализация классов 11](#_Toc89630294)

[5.3. Разработка тестового приложения 16](#_Toc89630295)

[6. Тестирование 17](#_Toc89630296)

[7. Заключение 31](#_Toc89630297)

[8. Список использованных источников 31](#_Toc89630298)

[Приложение А 31](#_Toc89630299)

### **Введение**

Информационная система - это организационно-техническая система, предназначенная для выполнения информационно-вычислительных работ или предоставления информационно-вычислительных услуг, удовлетворяющих потребности системы управления и ее пользователей - управленческого персонала, внешних пользователей.

Автоматизированные информационные системы стали играть в жизни общества большую роль в середине 20 века. В это время бурно начала развиваться индустрия вычислительной техники. Большинство компаний и организаций переходили на автоматизированные системы, потому что человеку становилось все сложнее управлять большим количеством информации и организовать ее должным образом. С этого момента у каждой компании появились свои информационные системы для стабильной работы.

В моём случае, я разрабатываю автоматизированную информационную систему «Больница» на языке программирования С++. Эта простая программа обеспечивает стабильную работу стационарной больницы, упрощает запись пациентов и врачей, ведение учета о заболеваниях отдельных пациентов, подсчет количества свободных мест в отделениях.

### **Задание на курсовое проектирование**

Вариант 17

Разработка системы классов для обеспечения работы больницы (В системе должны поддерживаться режимы учета больных по отделениям, заболеваниям, сложности заболевания и количеству заболеваний у одного человека, продолжительности болезни, количеству койко-мест и анализа заболеваемости по районам и категориям).

### **Постановка задачи**

### **Анализ предметной области**

Предметной областью курсового проекта является информационная система «Больница». Система должна реализовывать функции записи врачей и пациентов, добавление нового района и отделения, удаления всего вышеперечисленного, прикрепления пациентов к отделениям и врачам, подсчет свободных мест в отделениях и количества заболеваний у одного пациента, а также количество заболеваемых по районам.

### **Формулировка технического задания**

В результате анализа предметной области можно сформулировать следующие функциональные требования:

Программа должна быть реализована на языке С++ в виде приложения с графическим интерфейсом. Должно быть реализовано меню пользователя в виде кнопок, по нажатию на которые пользователь будет переходить в категории: «Пациенты», «Отделения», «Районы» и «Врачи». Реализованы следующие функции для работы с данными: добавление, удаление записи, вывод на экран.

**Для категории «Врачи» должны быть реализованы следующие функции:**

**Регистрация нового врача**

Функция выполняет добавление записи в список. При вводе данных выполняется проверка формата введенных данных. При вводе номера полиса обязательного медицинского страхования проверяется длина строки (должна быть равна 16 символам). Выполняется проверка на уникальность номера (два человека с одинаковым номером существовать не могут). Отделение больницы, к которому прикрепляется врач, выбирается пользователем из предложенного списка.

**Удаление записи**

Функция удаляет запись по введенному порядковому номеру, который отображается в списке со всеми врачами. Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр пациентов врача**

Функция показывает список пациентов, прикрепленных к врачу, который выбирается по порядковому номеру из списка врачей, выведенному на экран. Если порядковый номер не найден, выводится соответствующее сообщение на экран.

**Вывод на экран всех врачей**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.1).

Содержание столбцов таблицы

таблица 3.2.1

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Отделение | Должность | Номер полиса |

**Вывод на экран всех пациентов одного врача**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.2).

Содержание столбцов таблицы

таблица 3.2.2

|  |  |
| --- | --- |
| ФИО | Количество заболеваний |

**Для категории «Районы» должны быть реализованы следующие функции:**

**Добавление записи**

Функция выполняет добавление записи в список с введенным наименованием района.

**Удаление записи**

Функция удаляет запись по введенному порядковому номеру, который отображается в списке со всеми районами. Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр пациентов района**

Функция показывает список пациентов, прикрепленных к району, который выбирается по порядковому номеру из списка районов, выведенному на экран. Если порядковый номер не найден, выводится соответствующее сообщение на экран.

**Вывод на экран всех районов**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.3).

Содержание столбцов таблицы

таблица 3.2.3

|  |  |  |
| --- | --- | --- |
| Наименование | Количество зарегистрированных пациентов | Количество пациентов, находящихся в стационаре |

**Вывод на экран всех пациентов одного района**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.4).

Содержание столбцов таблицы

таблица 3.2.4

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Отделение | Количество заболеваний | Номер полиса |

**Для категории «Отделения» должны быть реализованы следующие функции:**

**Добавление записи**

Функция выполняет добавление записи в список. При вводе данных выполняется проверка формата введенных данных. При вводе количества койко-мест число проверяется на отрицательность и равность нулю (неверный ввод). Выполняется проверка на уникальность наименования отделения, которое выбирается из предложенного списка (два отделения с одинаковым наименованием существовать не могут).

**Удаление записи**

Функция удаляет запись по введенному порядковому номеру, который отображается в списке со всеми отделениями. Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр пациентов и врачей отделения**

Функция показывает список пациентов и врачей, прикрепленных к отделению, который выбирается по порядковому номеру из списка с отделениями, выведенному на экран. Если порядковый номер не найден, выводится соответствующее сообщение на экран.

**Вывод на экран всех отделений**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.5).

Содержание столбцов таблицы

таблица 3.2.5

|  |  |  |
| --- | --- | --- |
| Наименование | Количество койко-мест | Количество свободных мест |

**Вывод на экран всех пациентов и врачей одного отделения**

Функция выводит на экран содержимое списков в виде таблиц (таблица 3.2.6, таблица 3.2.7).

Содержание столбцов таблицы с пациентами

таблица 3.2.6

|  |  |  |
| --- | --- | --- |
| ФИО | Количество заболеваний | Номер полиса |

Содержание столбцов таблицы с врачами

таблица 3.2.7

|  |  |
| --- | --- |
| ФИО | Должность |

**Для категории «Пациенты» должны быть реализованы следующие функции:**

**Регистрация нового пациента**

Функция выполняет добавление записи в список. При вводе данных выполняется проверка формата введенных данных. При вводе номера полиса обязательного медицинского страхования проверяется длина строки (должна быть равна 16 символам). Выполняется проверка на уникальность номера (два человека с одинаковым номером существовать не могут). Отделение больницы, к которому прикрепляется пациент, выбирается пользователем из предложенного списка.

**Удаление записи**

Функция удаляет запись по введенному порядковому номеру, который отображается в списке со всеми пациентами. Если такая запись не найдена, выводится соответствующее сообщение на экран.

**Просмотр заболеваний и врачей пациента**

Функция показывает список заболеваний и прикрепленных врачей пациента, который выбирается по порядковому номеру из списка с пациентами, выведенному на экран. Если порядковый номер не найден, выводится соответствующее сообщение на экран.

**Вывод на экран всех пациентов**

Функция выводит на экран содержимое списка в виде таблицы (таблица 3.2.8).

Содержание столбцов таблицы

таблица 3.2.8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ФИО | Район | Отделение | Количество заболеваний | Номер полиса |

**Вывод на экран всех заболеваний и врачей одного пациента**

Функция выводит на экран содержимое списков в виде таблиц (таблица 3.2.9, таблица 3.2.10).

Содержание столбцов таблицы с заболеваниями

таблица 3.2.9

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Тяжесть | Начало | Окончание |

Содержание столбцов таблицы с врачами

таблица 3.2.10

|  |  |
| --- | --- |
| ФИО | Должность |

**Выписка пациента**

Функция устанавливает дату окончания болезни и открепляет пациента от отделения. Для этого необходимо ввести порядковый номер, который отображается в списке со всеми пациентами, и порядковый номер болезни, который отображается в списке со всеми болезнями пациента, дата выбирается в специальном виджете. Если порядковые номера введены неверно, выводится соответствующее сообщение на экран.

**Прикрепление пациента к отделению**

Функция выполняет добавление в список пациентов отделения больного, выбранного по порядковому номеру из списка на экране. Необходимо ввести также порядковый номер врача, который отображается в списке со всеми врачами отделения. Отделение выбирается из предложенного выпадающего списка. Если порядковые номера введены неверно, выводится соответствующее сообщение на экран. Также должно быть указано количество койко-мест. Тяжесть заболевания выбирается из выпадающего списка, дата начала болезни указывается с помощью специального виджета.

### **Проектирование классов**

### **Разработанные классы**

Классы сущностей:

District – класс сущности район

Disease – класс сущности болезнь

Doctor – класс сущности врач

Human – класс сущности человек

Patient – класс сущности пациент

Ward – класс сущности отделение

Интерфейсные классы:

MainWindow – класс главного меню

Dialog\_district – класс диалогового окна для категории «Районы»

Dialog\_doctor – класс диалогового окна для категории «Врачи»

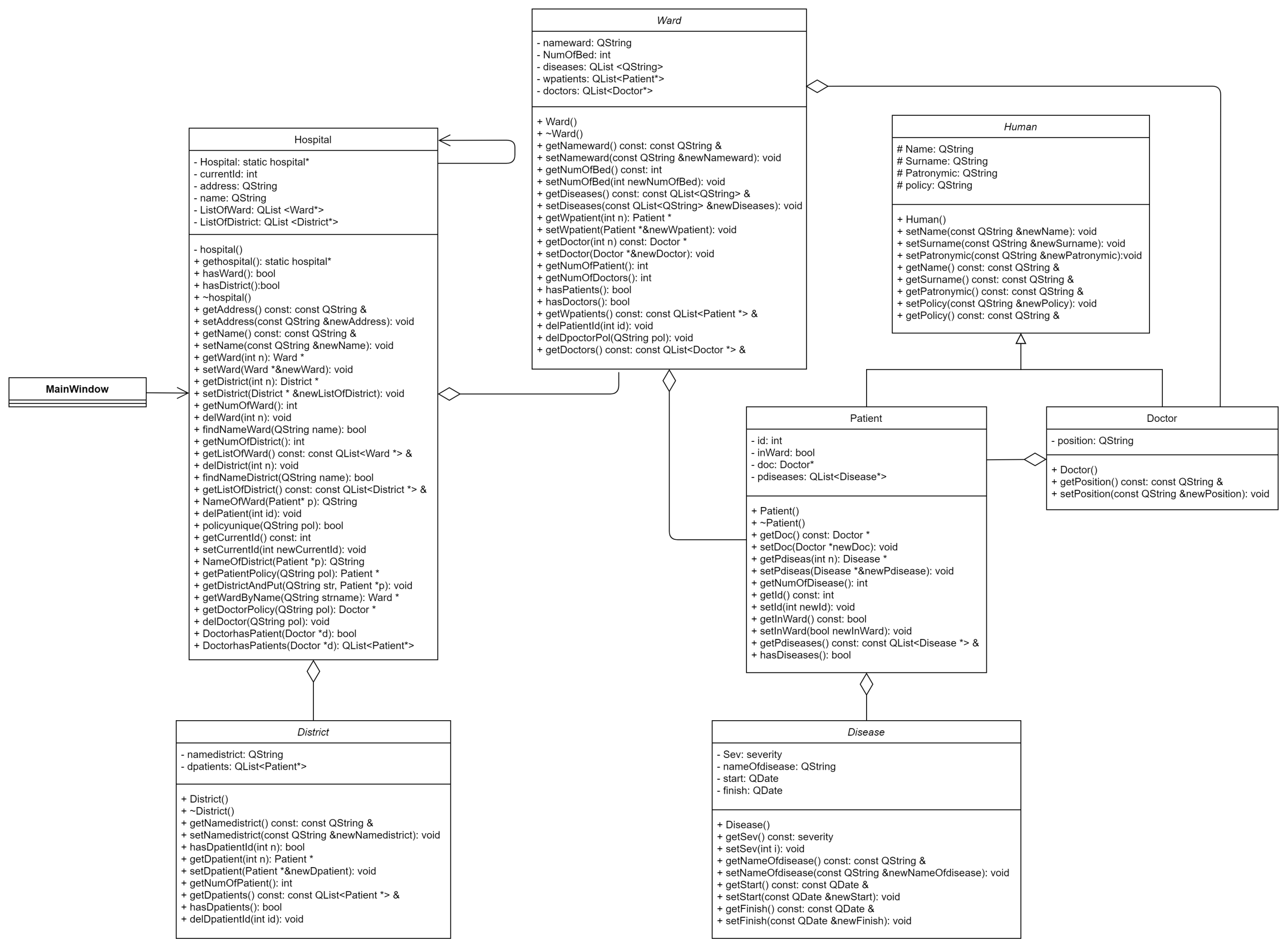
Dialog\_patient – класс диалогового окна для категории «Пациенты»

Dialog\_ward – класс диалогового окна для категории «Отделения»

Классы управления:

Hospital – класс управления (больница), он же является классом паттерна Singleton

### **Диаграмма классов**



### **Разработка приложения**

### **Разработка интерфейса приложения**

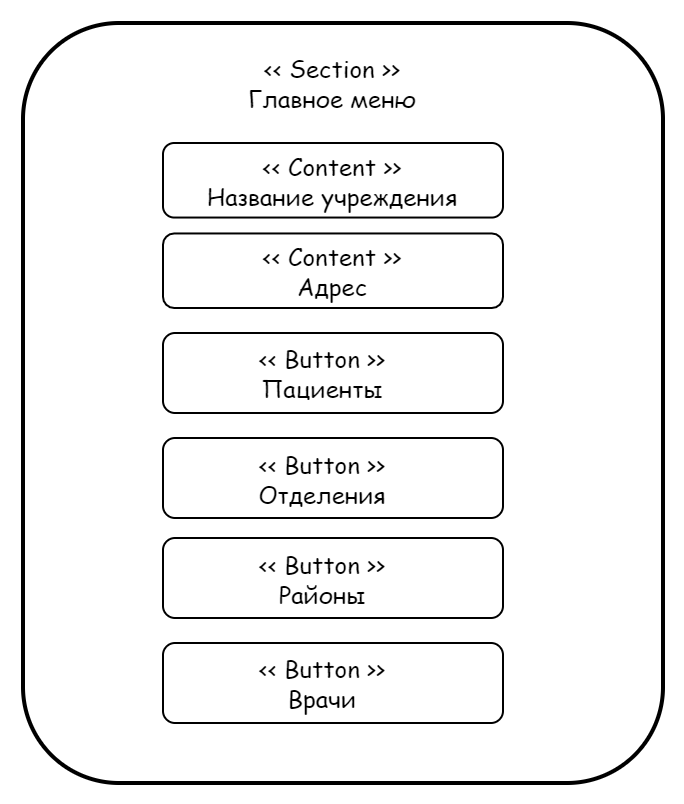


Рис. 5.1.1. Интерфейс главного меню

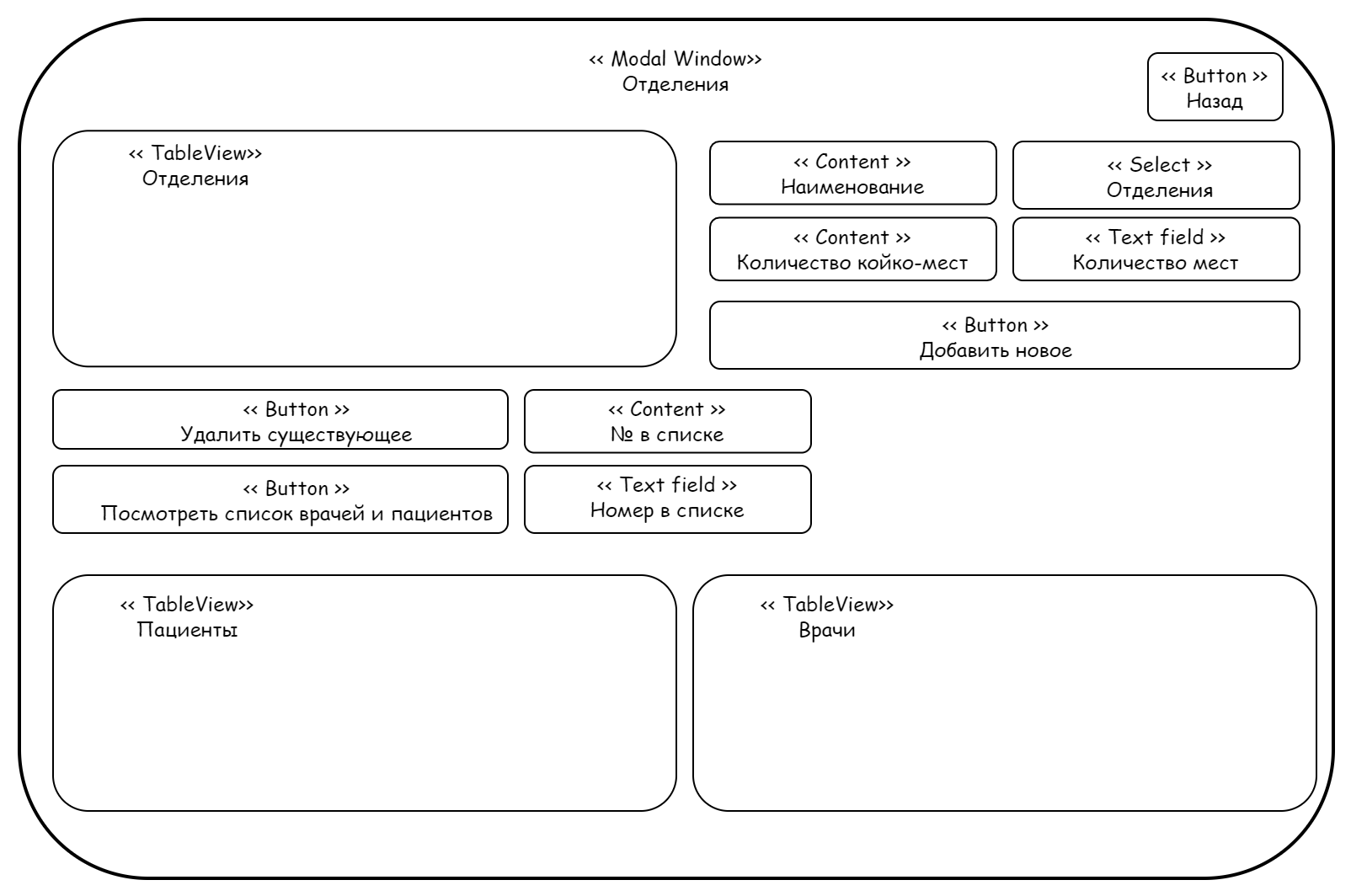


Рис. 5.1.2. Интерфейс модального окна с функциями для категории «отделения больницы»

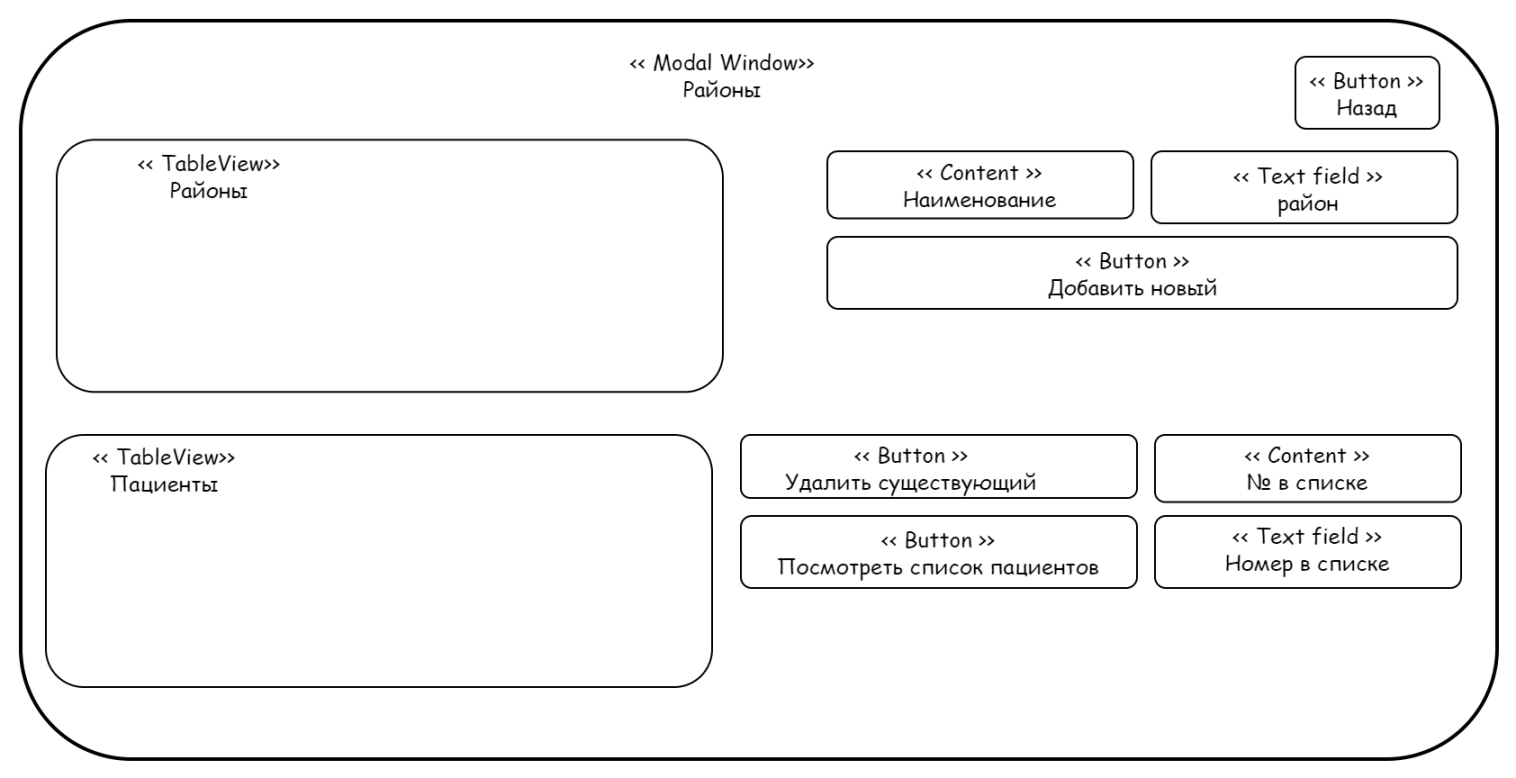


Рис. 5.1.3. Интерфейс модального окна с функциями для категории «районы»

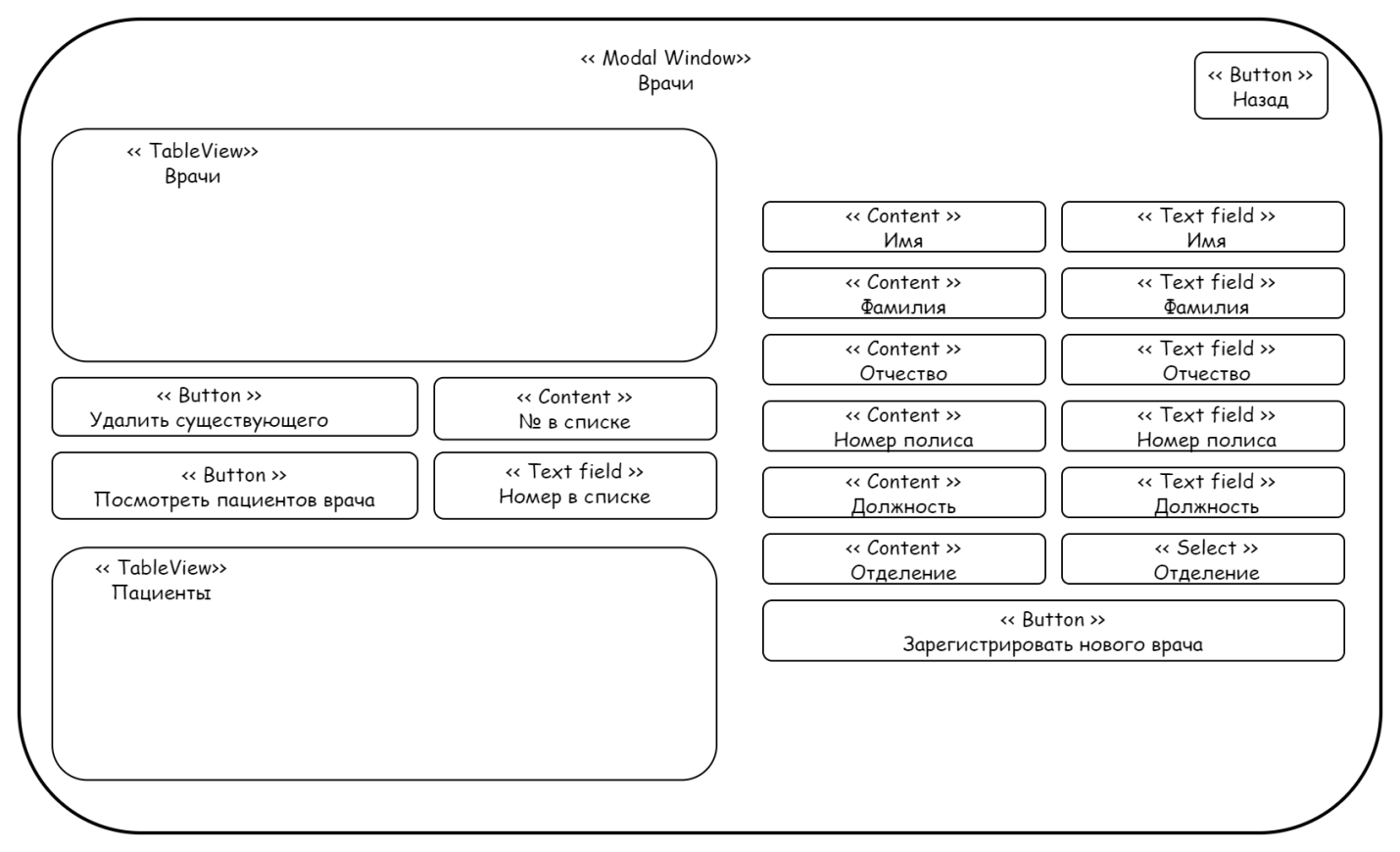


Рис. 5.1.4. Интерфейс модального окна с функциями для категории «врачи»

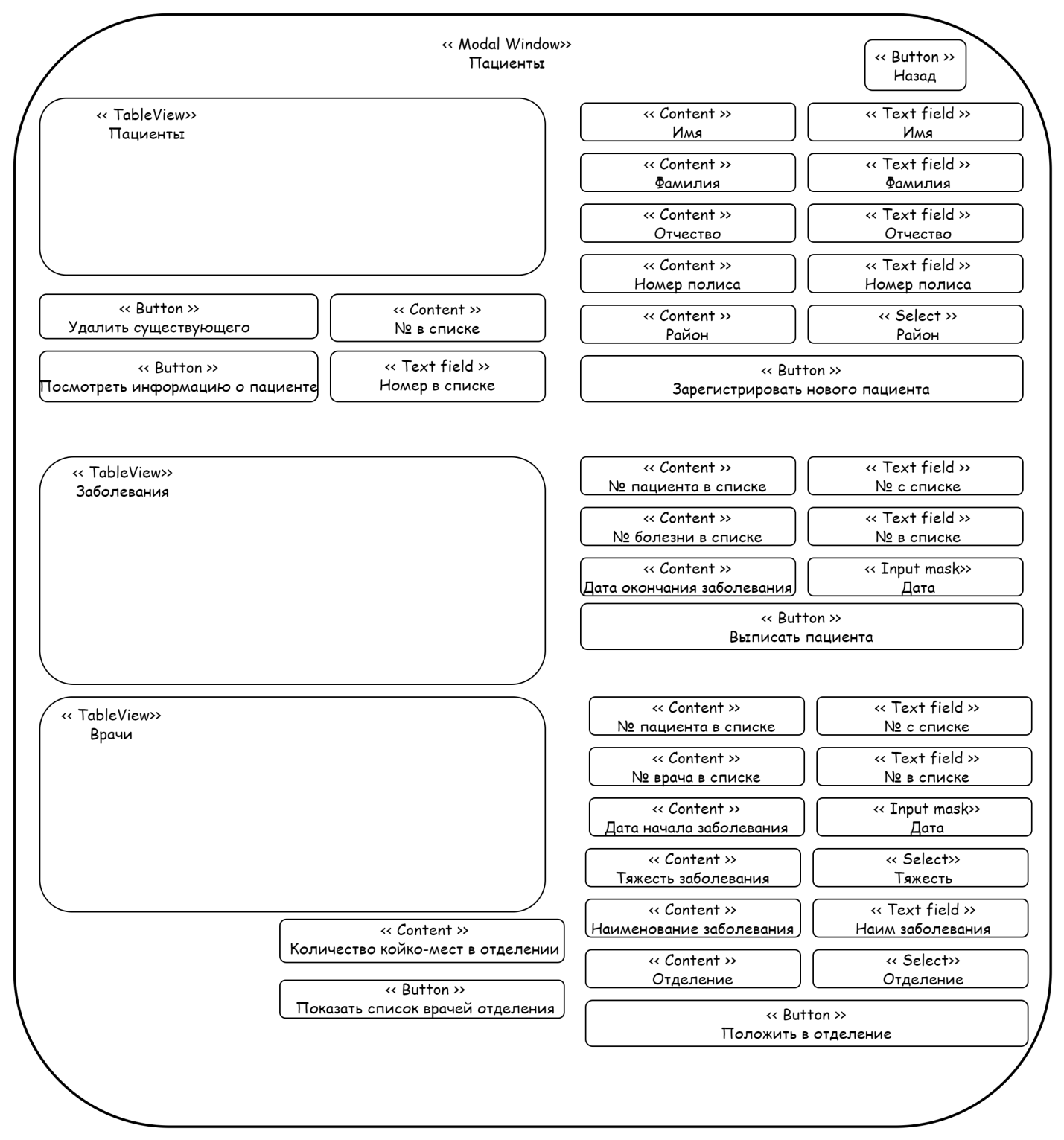


Рис. 5.1.5. Интерфейс модального окна с функциями для категории «пациенты»

### **Реализация классов**

Реализация класса сущности «район»:

class District

{

public:

District();

~District();

//сеттеры и геттеры для хранимых данных

const QString &getNamedistrict() const;

void setNamedistrict(const QString &newNamedistrict);

Patient \*getDpatient(int n);

void setDpatient(Patient \*&newDpatient);

const QList<Patient \*> &getDpatients() const;

//проверка есть ли пациент в списке по Id

bool hasDpatientId(int n);

//получение числа пациентов в списке

int getNumOfPatient();

//проверка есть ли пациенты в списке

bool hasDpatients();

//удаление пациента по Id

void delDpatientId(int id);

private:

QString namedistrict; //название района

QList<Patient\*> dpatients; //пациенты

//в список пациентов будут заноситься/регистрироваться паценты, которые могут не лежать в отделении

};

Реализация класса сущности «заболевания»:

class Disease

{

public:

//степень тяжести болезни

enum severity{

light,

medium,

heavy

};

Disease();

//сеттеры и геттеры для хранимых данных

severity getSev() const;

void setSev(int i);

const QString &getNameOfdisease() const;

void setNameOfdisease(const QString &newNameOfdisease);

const QDate &getStart() const;

void setStart(const QDate &newStart);

const QDate &getFinish() const;

void setFinish(const QDate &newFinish);

private:

severity Sev; //тяжесть болезни

QString nameOfdisease; //название

QDate start; //дата начала болезни

QDate finish; //дата окончания

};

Реализация класса сущности «человек»:

class Human

{

public:

Human();

//сеттеры и геттеры для хранимых данных

void setName(const QString &newName);

void setSurname(const QString &newSurname);

void setPatronymic(const QString &newPatronymic);

const QString &getName() const;

const QString &getSurname() const;

const QString &getPatronymic() const;

void setPolicy(const QString &newPolicy);

const QString &getPolicy() const;

protected:

QString Name; //имя

QString Surname;//фамилия

QString Patronymic;//отчество

QString policy; //номер полиса медицинского страхования

};

Реализация класса сущности «врач»:

class Doctor : public Human

{

public:

Doctor();

//сеттер и геттер для хранимого данного

const QString &getPosition() const;

void setPosition(const QString &newPosition);

private:

QString position; //должность

};

Реализация класса сущности «пациент»:

class Patient: public Human

{

public:

Patient();

~Patient();

//сеттеры и геттеры для хранимых данных

Doctor \*getDoc() const;

void setDoc(Doctor \*newDoc);

const QList<Disease \*> &getPdiseases() const;

void setPdiseas(Disease \*&newPdisease);

int getId() const;

void setId(int newId);

bool getInWard() const;

void setInWard(bool newInWard);

//получение числа заболеваний

int getNumOfDisease();

//получение болезни по порядковому номеру в списке

Disease \*getPdiseas(int n);

//проверка есть ли болезни в списке

bool hasDiseases();

private:

int id; //идентификатор

bool inWard=0; //находится в отделении или нет

Doctor\* doc; //прикрепленный врач

QList<Disease\*> pdiseases;//список болезней

//список болезней представляет собой импровизированную амбулаторную карту

//в ней хранится информация о истории болезней

};

Реализация класса сущности «отделение»:

class Ward

{

public:

Ward();

~Ward();

//сеттеры и геттеры для хранимых данных

const QString &getNameward() const;

void setNameward(const QString &newNameward);

int getNumOfBed() const;

void setNumOfBed(int newNumOfBed);

const QList<QString> &getDiseases() const;

void setDiseases(const QList<QString> &newDiseases);

Patient \*getWpatient(int n);

void setWpatient(Patient \*&newWpatient);

Doctor \*getDoctor(int n) const;

void setDoctor(Doctor \*&newDoctor);

const QList<Patient \*> &getWpatients() const;

const QList<Doctor \*> &getDoctors() const;

//получение количества пациентов

int getNumOfPatient();

//получение количества врачей

int getNumOfDoctors();

//проверка есть ли пациенты в списке

bool hasPatients();

//проверка есть ли врачи в списке

bool hasDoctors();

//удаление пациента по Id

void delPatientId(int id);

//удаление врача по номеру полиса

void delDpoctorPol(QString pol);

private:

QString nameward; //название отделения

int NumOfBed; //количество коек

QList <QString> diseases; //список болезней с которыми кладут в отделение

QList<Patient\*> wpatients; //пациенты

QList<Doctor\*> doctors; //доктора

};

Реализация класса управления (паттерн Singleton):

class hospital

{

public:

static hospital\* gethospital();

//проверка есть ли отделения

bool hasWard();

//проверка есть ли районы

bool hasDistrict();

~hospital();

//геттеры и сеттеры для хранимых данных

const QString &getAddress() const;

void setAddress(const QString &newAddress);

const QString &getName() const;

void setName(const QString &newName);

Ward \*getWard(int n) ;

const QList<Ward \*> &getListOfWard() const;

void setWard(Ward \*&newWard);

District \* getDistrict(int n) ;

const QList<District \*> &getListOfDistrict() const;

void setDistrict(District \* &newListOfDistrict);

int getCurrentId() const;

void setCurrentId(int newCurrentId);

int getNumOfWard();

//удаление отделения по индексу

void delWard(int n);

//проверка есть ли отделение с таким именем

bool findNameWard(QString name);

//получение количества районов

int getNumOfDistrict();

//удаление района по индексу

void delDistrict(int n);

//проверка есть ли район с таким именем

bool findNameDistrict(QString name);

//получение названия отделения по пациенту

QString NameOfWard(Patient\* p);

//удаление пациента по Id

void delPatient(int id);

//проверка является ли полис уникальным среди людей

bool policyunique(QString pol);

//получение названия района по пациенту

QString NameOfDistrict(Patient \*p);

//получение пациента по полису

Patient \*getPatientPolicy(QString pol);

//добавление пациента в район по названию района

void getDistrictAndPut(QString str, Patient \*p);

//получение отделения по названию

Ward \*getWardByName(QString strname);

//получение врача по полису

Doctor \*getDoctorPolicy(QString pol);

//удаление врача по полису

void delDoctor(QString pol);

//проверка есть ли у доктора пациенты

bool DoctorhasPatient(Doctor \*d);

//получение списка пациентов по доктору

QList<Patient\*> DoctorhasPatients(Doctor \*d);

private:

hospital();

static hospital\* Hospital;

int currentId;

QString address; //адрес больницы

QString name; //название больницы

QList <Ward\*> ListOfWard; //список с отделениями

QList <District\*> ListOfDistrict;//список с районами

};

### **Разработка тестового приложения**

Рассмотрим алгоритмы сложных методов классов диалоговых окон:

Псевдокод метода вывода таблицы с районами класса Dialog\_district:

*Если есть районы в списке*

*В переменную n записываем количество районов*

*Устанавливаем в виджет таблицы n строк*

*Для каждого значения от 1 до n*

*Нумеруем строки*

*Создаем указатель на ячейку таблицы*

*Для каждой строки в первую колонку*

*Записываем название района*

*Для каждой строки во вторую колонку*

*Записываем количество пациентов в районе*

*Для каждой строки в третью колонку*

*Записываем количество заболевших*

*Выравниваем таблицу по ширине колонок*

Алгоритм методов вывода аналогичен для других таблиц приложения

Псевдокод метода удаления района из списка класса Dialog\_district:

*Если есть районы в списке*

*Считываем с виджета порядковый номер в переменную n*

*Инкрементирем n*

*Если n входит в пределы списка*

*Если район под данным номером не содержит пациентов*

*Удаляем район*

*Обновляем таблицу (показ)*

*Иначе*

*Выводим сообщение о том, что удаление невозможно*

*Иначе*

*Выводим сообщение о том, что такого района не существует*

Алгоритм методов удаления элемента из списка для других списков приложения аналогичен данному.

Псевдокод алгоритма метода добавления района в список класса Dialog\_distric:

*Считываем наименование нового района из виджета*

*Если район с таким наименованием уже зарегистрирован*

*Выводим сообщение о том, что такой район уже существует*

*Иначе если строка наименования не пуста*

*Создаем новый объект класса района*

*Устанавливаем значение поля наименование*

*Добавляем объект в список с районами*

*Выводим сообщение о том, что район успешно добавлен*

*Обновляем показ таблицы*

*Иначе*

*Выводим сообщение о том, что ввод неверен*

Алгоритм методов добавления элемента в список для других списков приложения аналогичен данному.

### **Тестирование**

*Диалоговое окно «Врачи»:*

**Регистрация нового врача:**

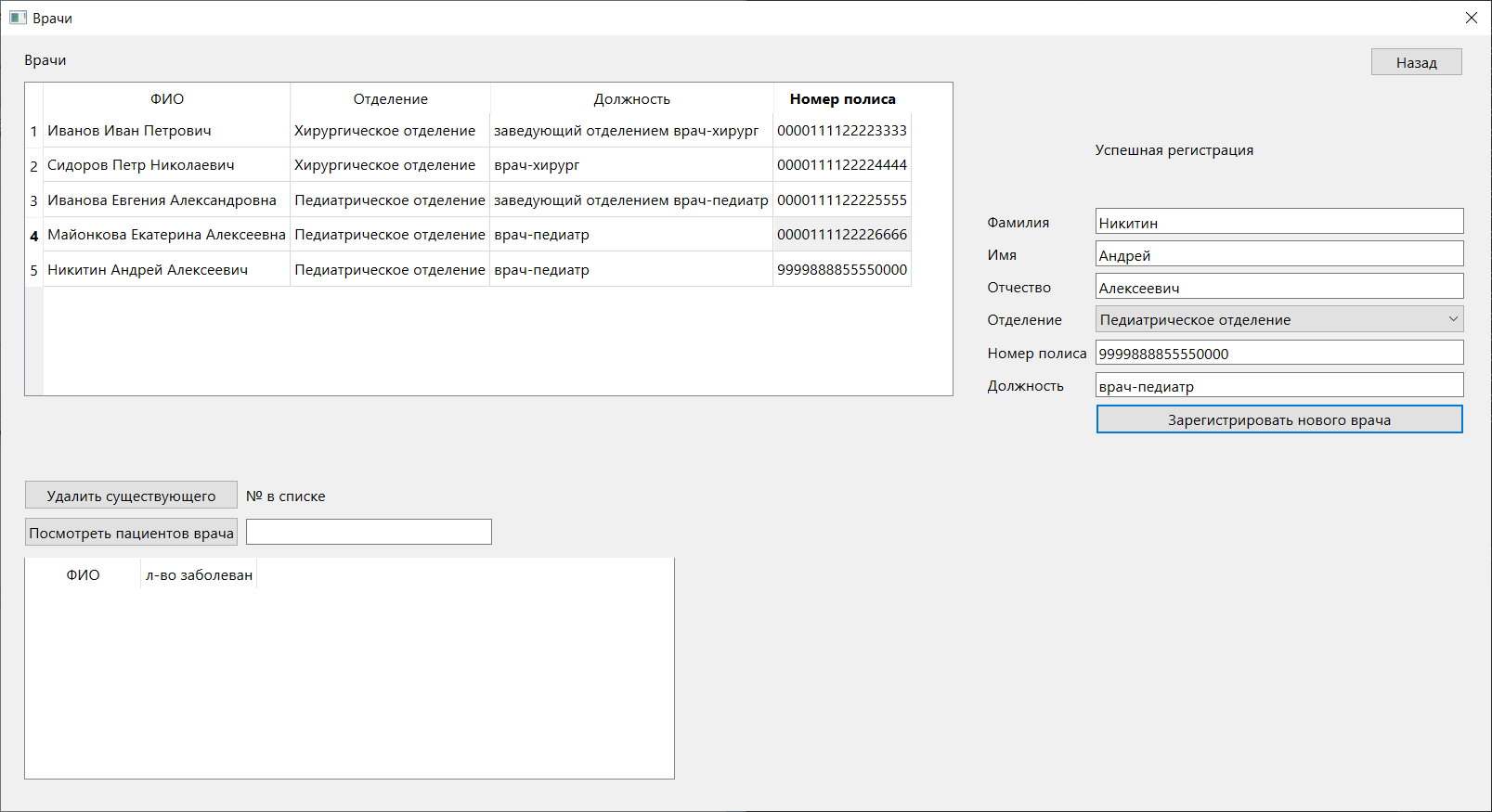


Рис. 6.1. Успешная регистрация врача

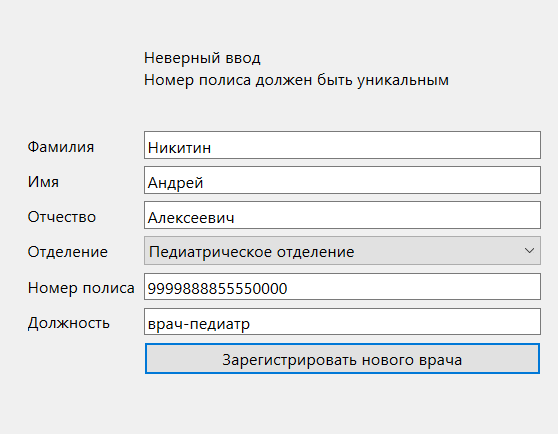


Рис. 6.2. Неверный ввод полиса при регистрации (человек с таким полисом существует)

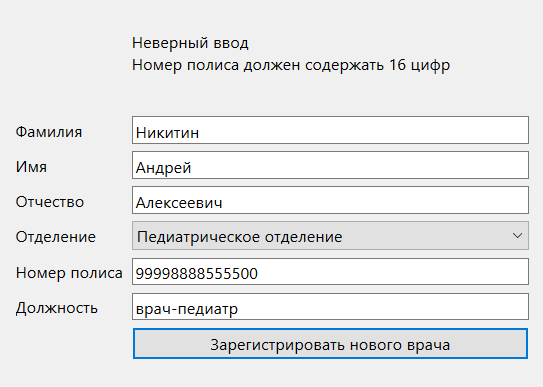


Рис. 6.2. Неверный ввод полиса при регистрации (меньше 16 цифр)

**Посмотреть пациентов врача:**

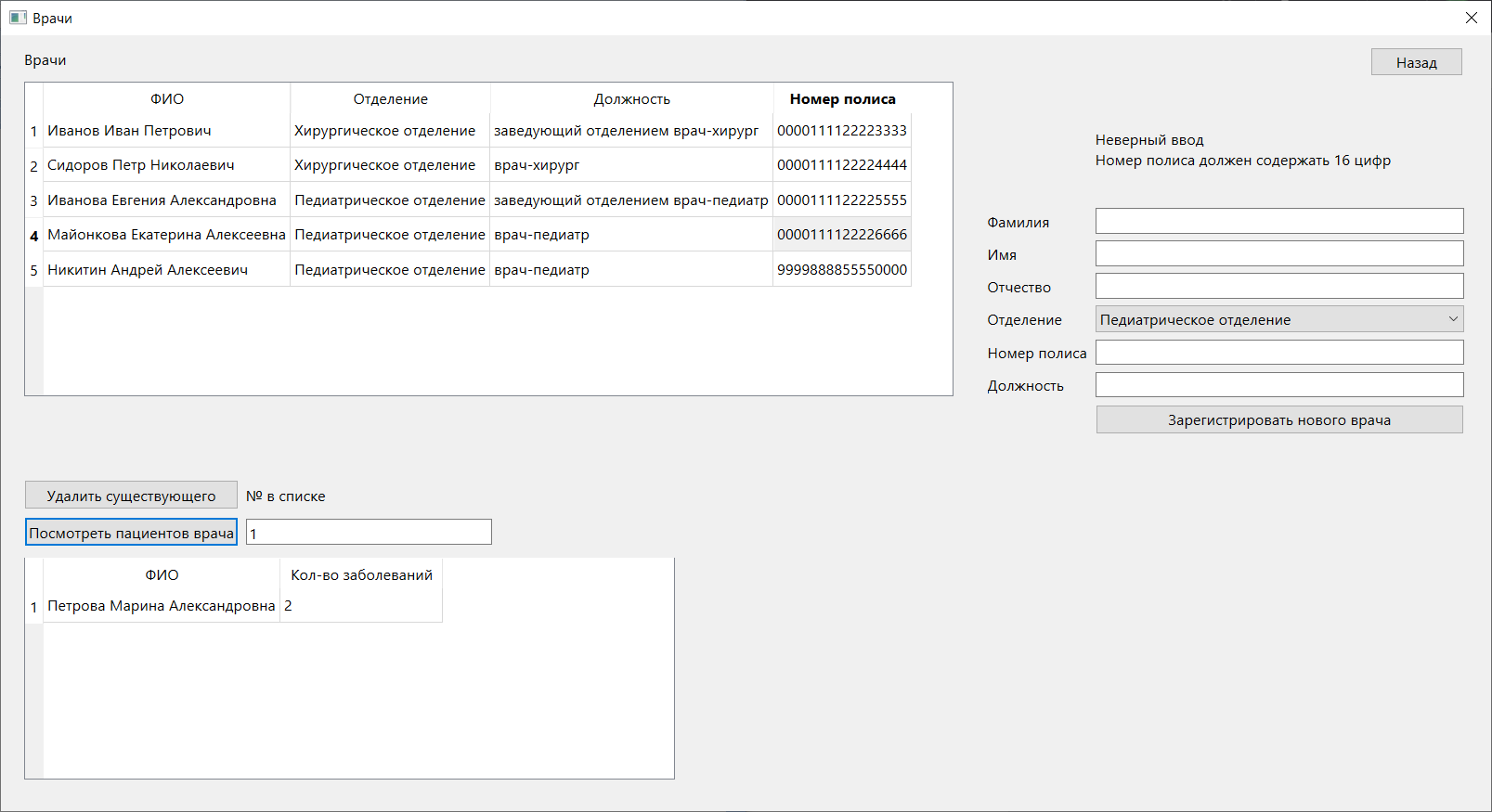


Рис. 6.4. Успешный просмотр пациентов врача

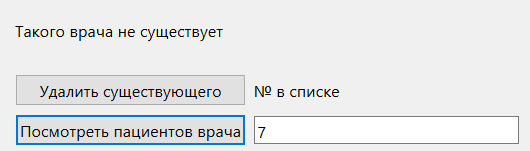


Рис. 6.5. Неверный ввод порядкового номера

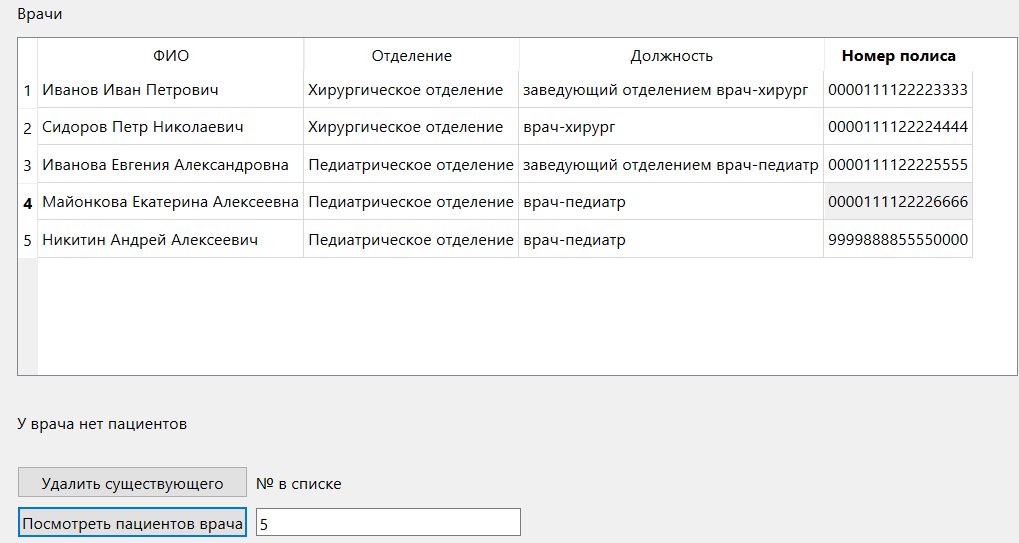


Рис. 6.6. Пустой список пациентов

**Удаление врача:**

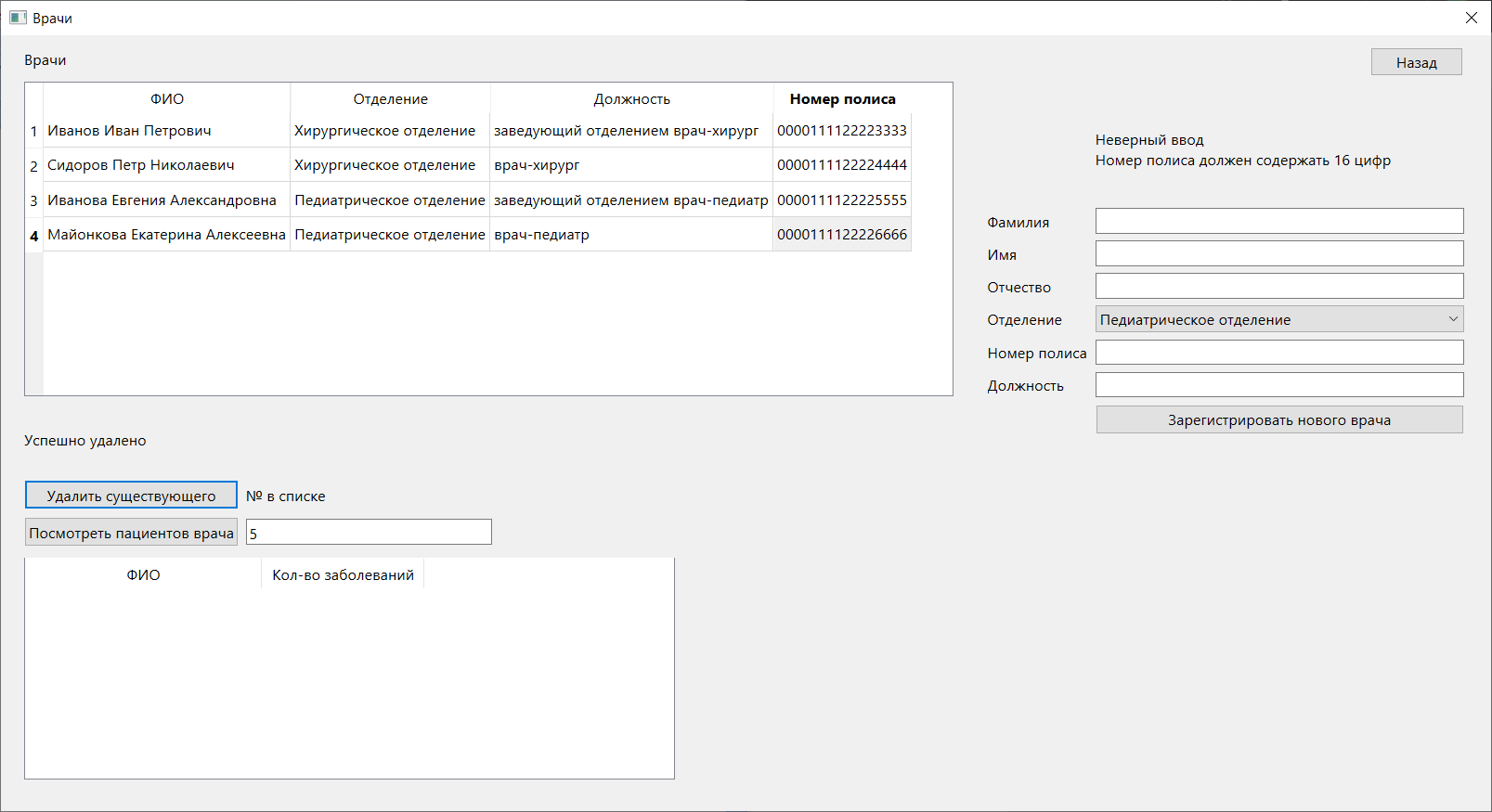


Рис. 6.7. Успешное удаление врача

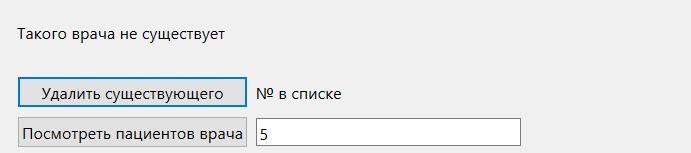


Рис. 6.8. Неверный ввод порядкового номера

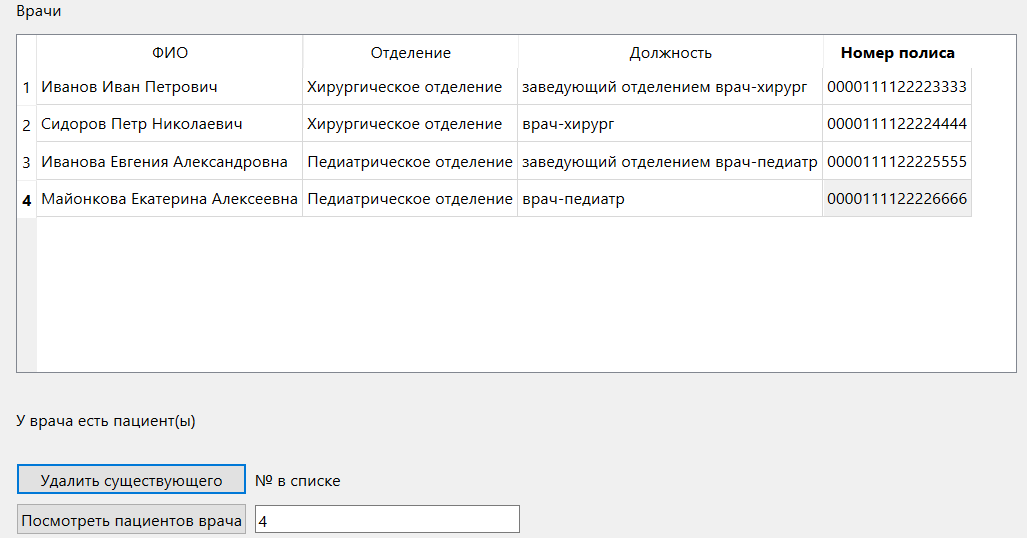


Рис. 6.9. Удаление врача, у которого есть пациенты

*Диалоговое окно «Районы»:*

**Добавить новый район:**

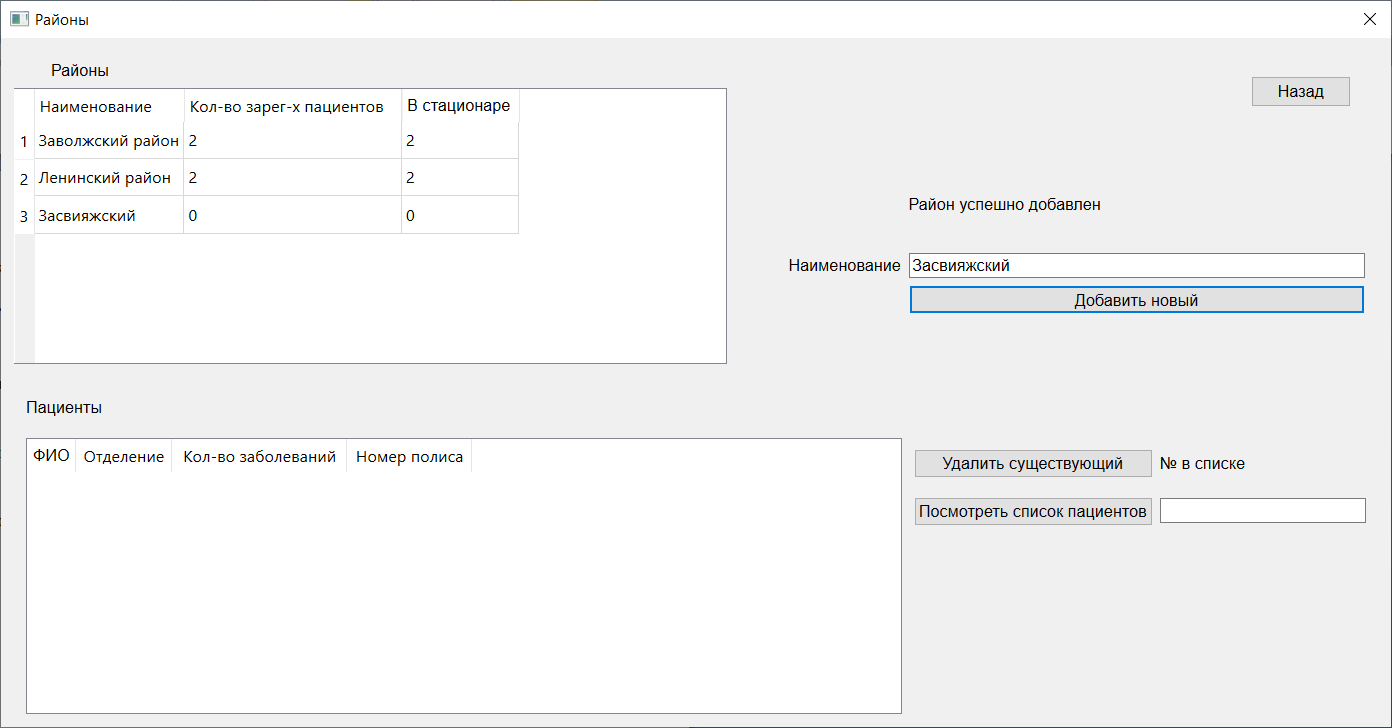


Рис. 6.10. Успешное добавление нового района

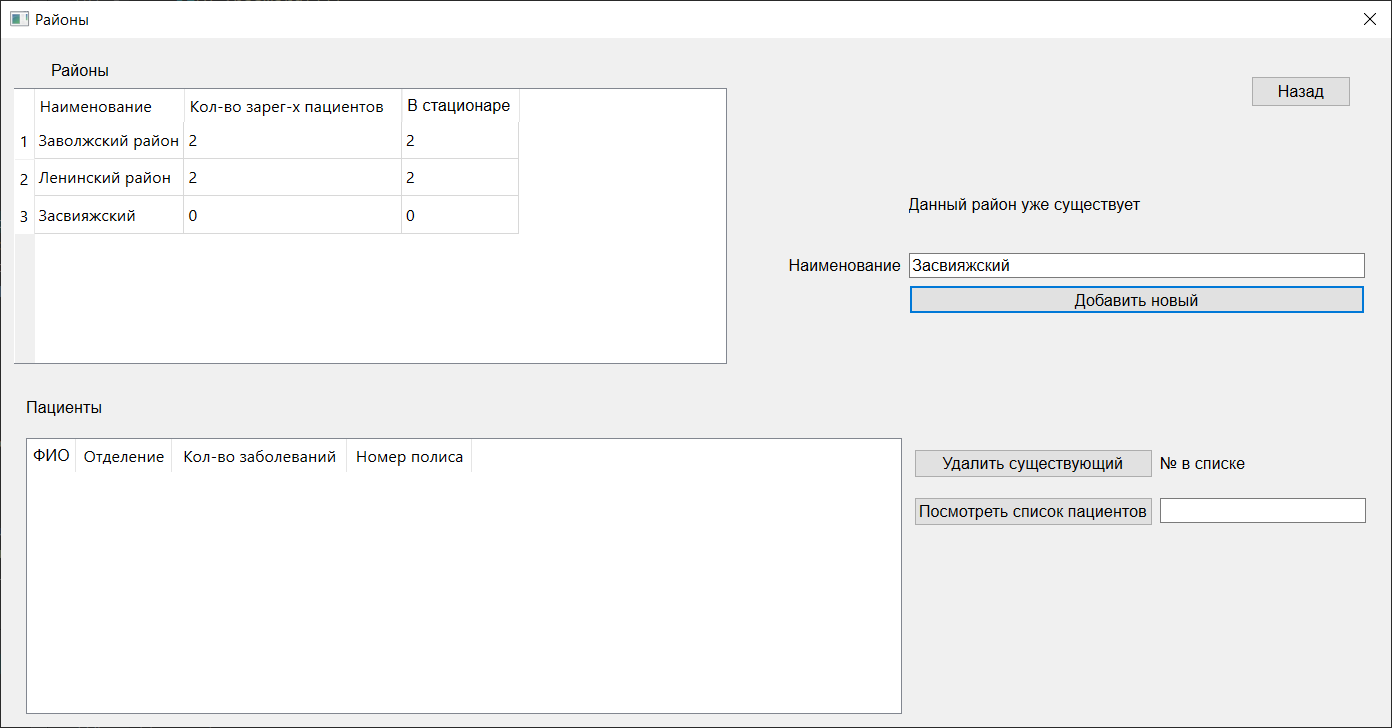


Рис. 6.11. Неверный ввод наименования (район уже существует)

**Посмотреть список пациентов:**

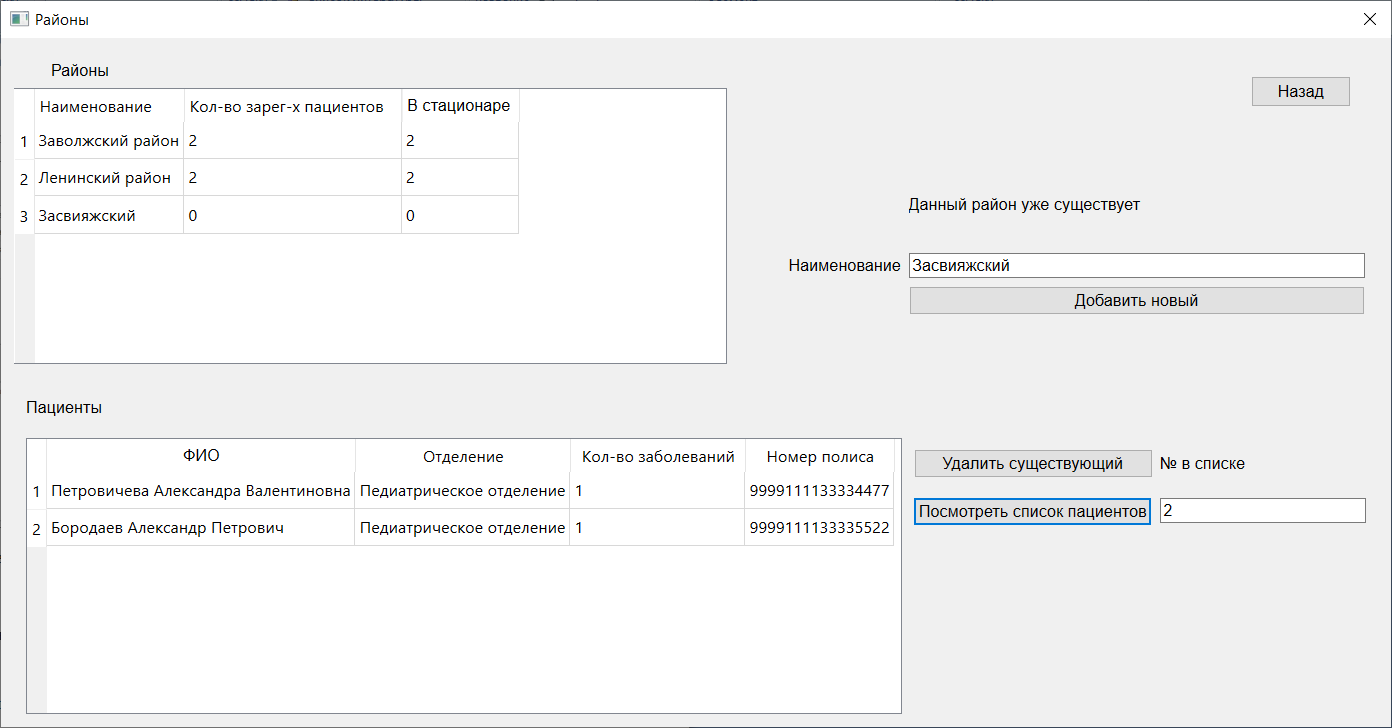
****

Рис. 6.12. Успешный просмотр списка пациентов

**Удаление района:**

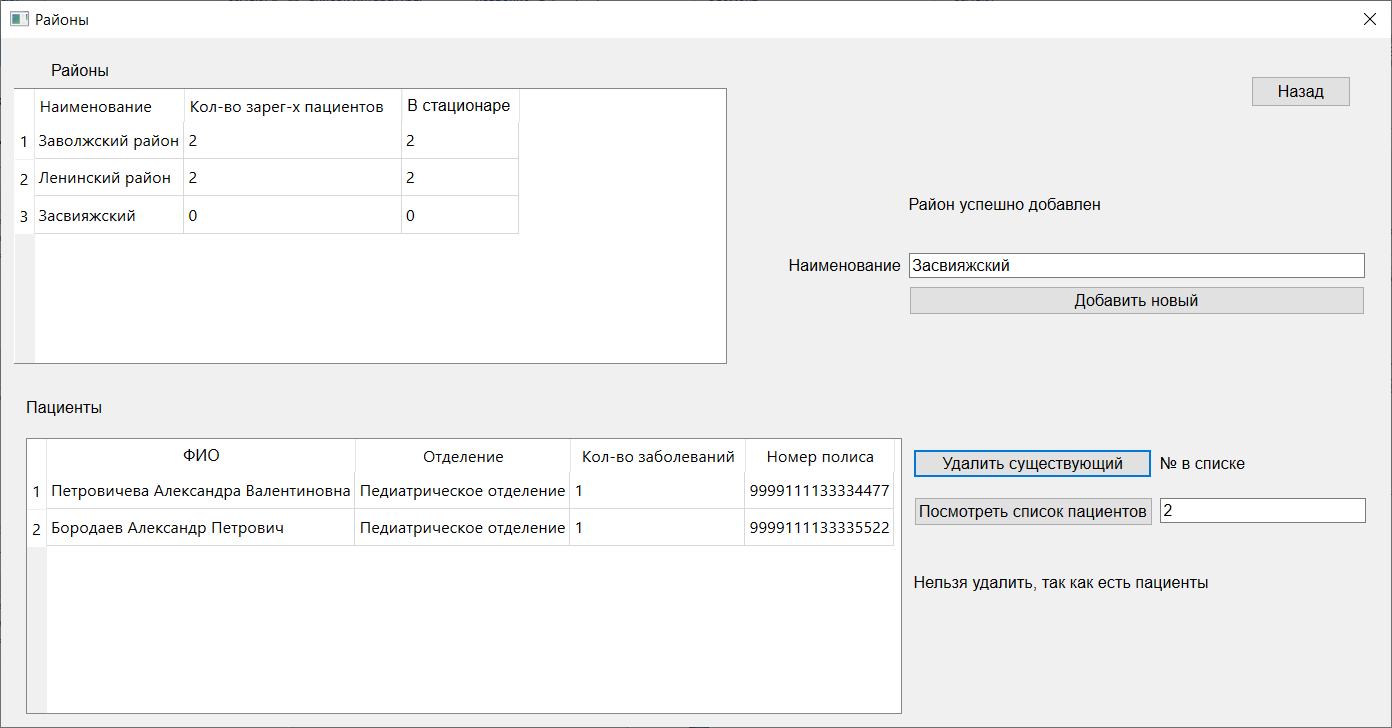
****

Рис. 6.13. Неверный ввод (в районе есть пациенты)

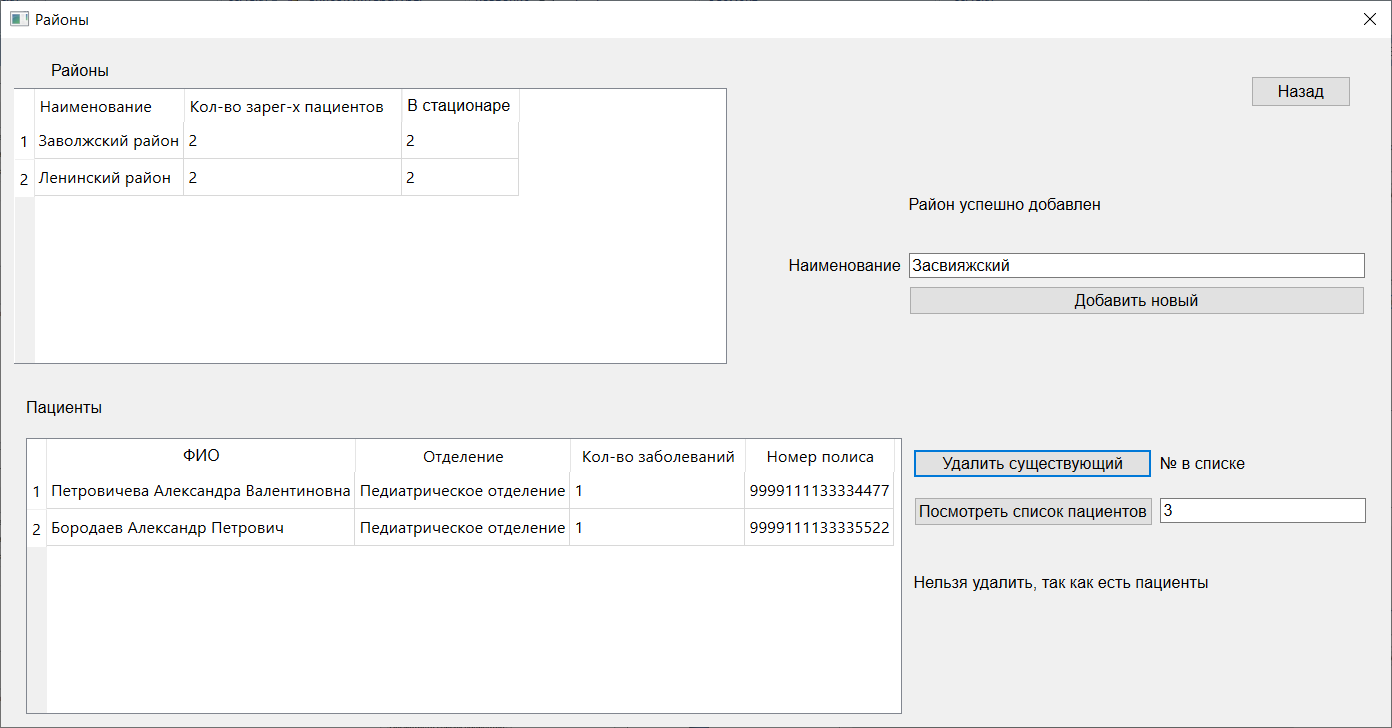
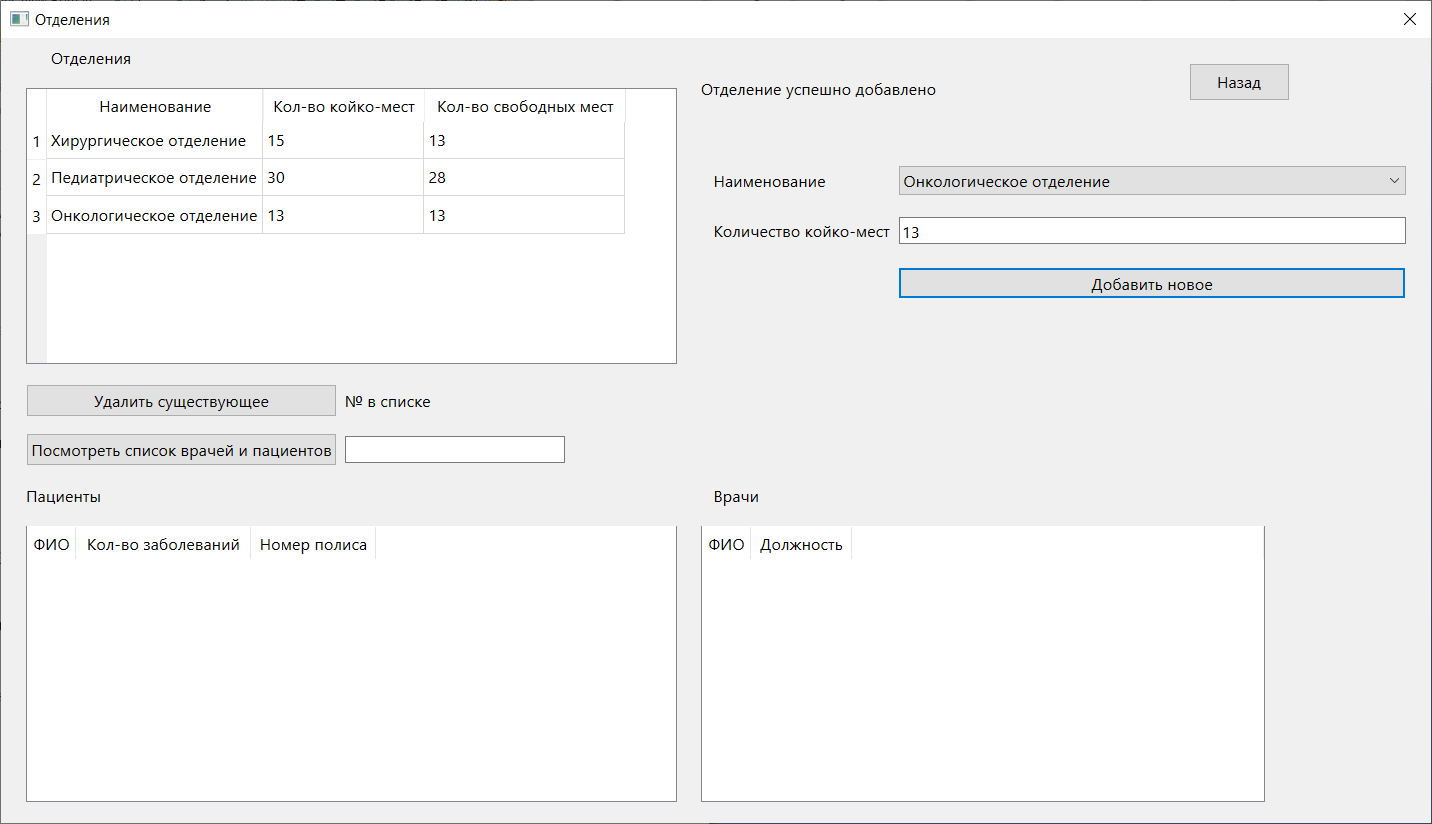
****

Рис. 6.14. Успешное удаление района

*Диалоговое окно «Отделения»:*

**Добавить новое отделение:**

****Рис. 6.15. Успешное добавление отделения

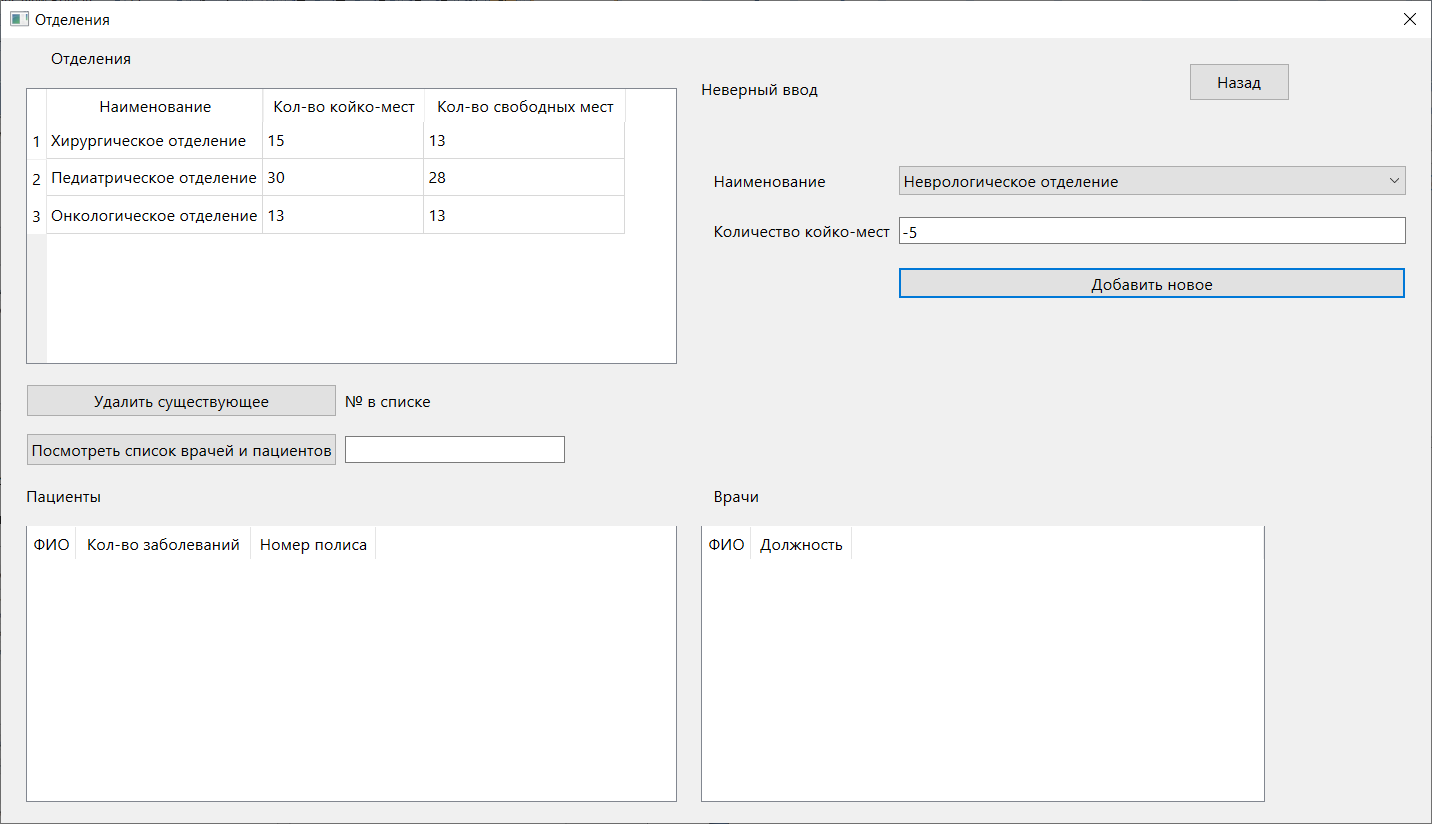


Рис. 6.16. Неверный ввод количества койко-мест (отрицательное число)

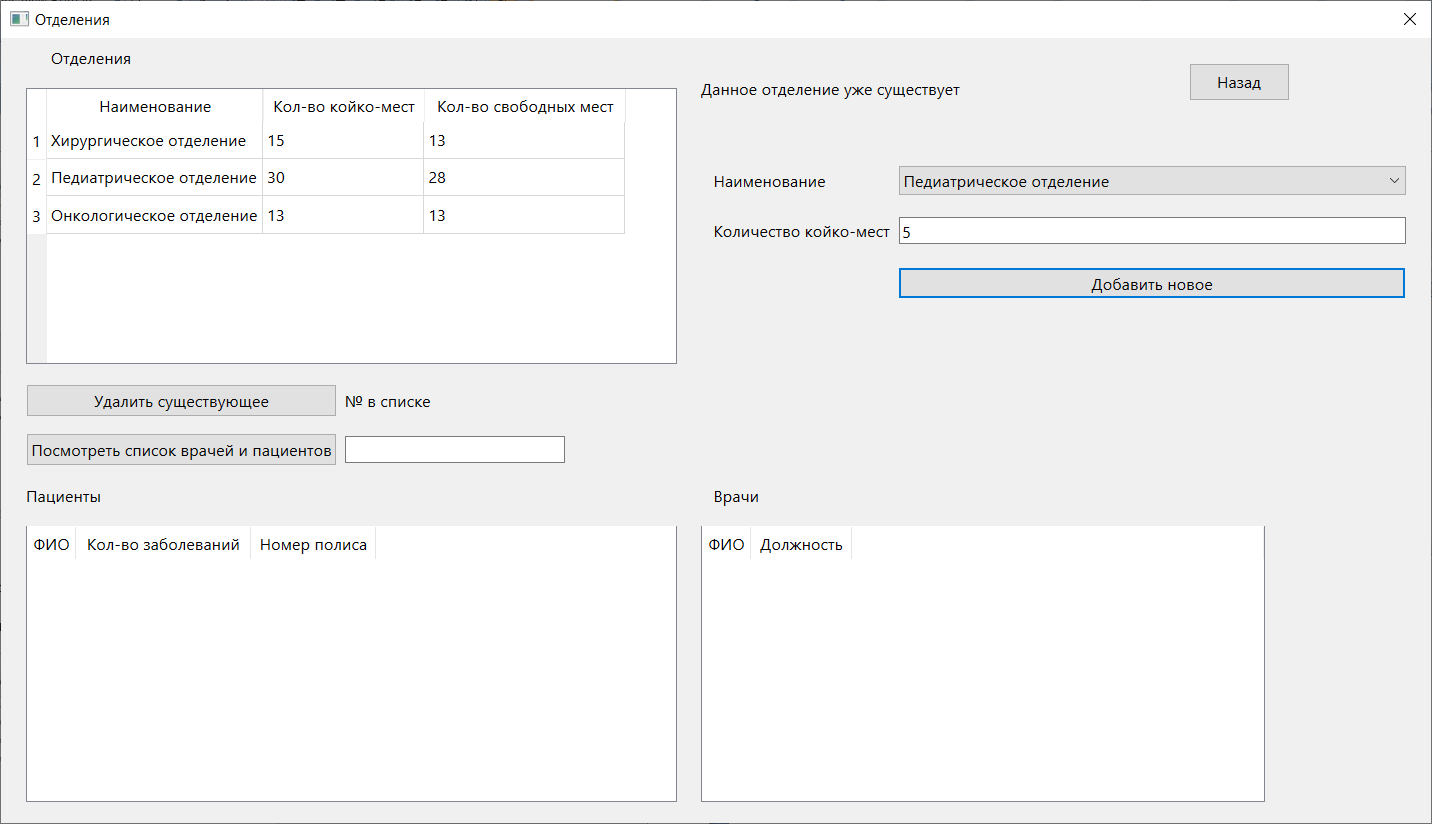


Рис. 6.17. Неверный ввод (отделение уже существует)

**Просмотр пациентов и врачей:**

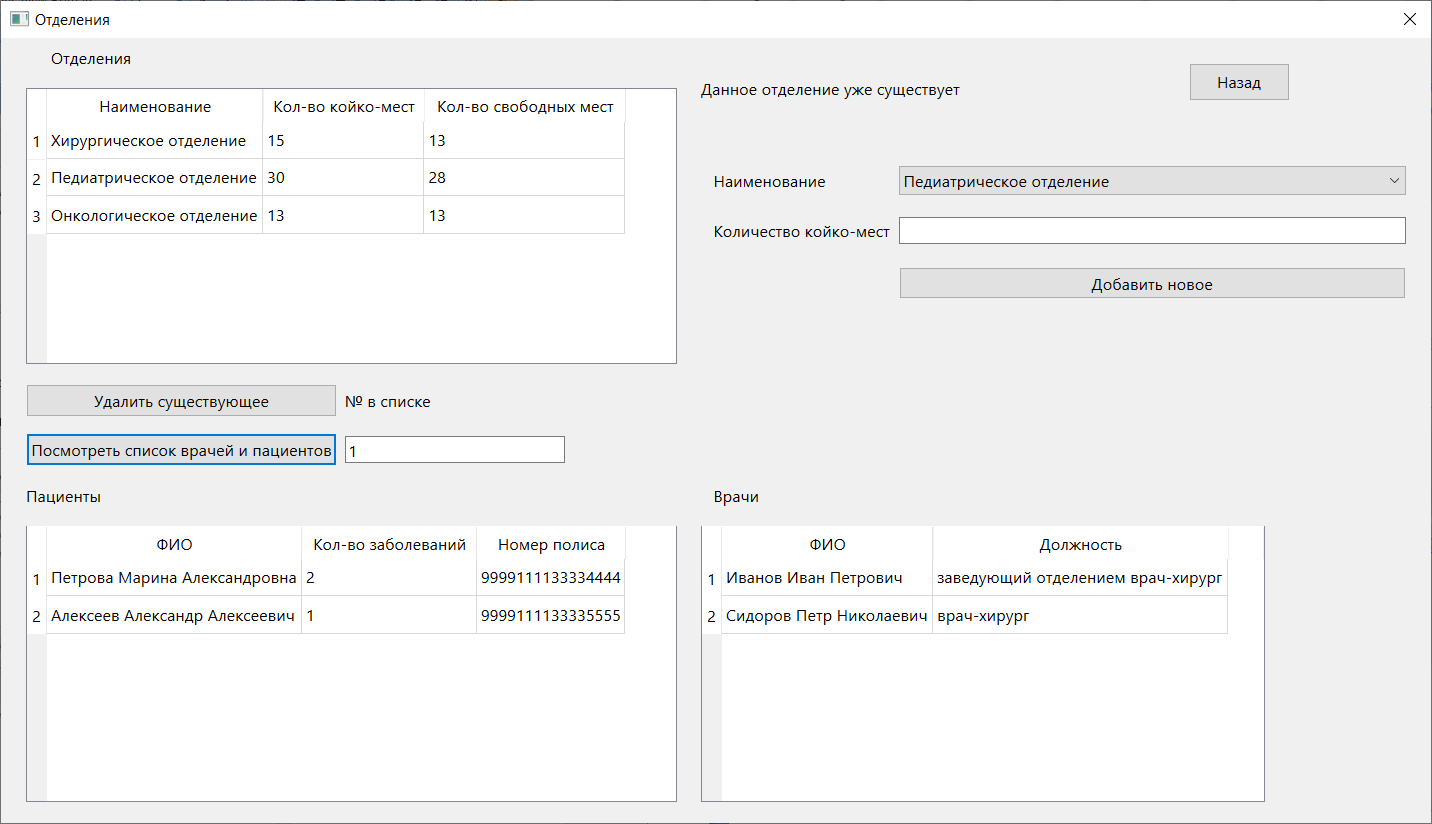


Рис. 6.18. Успешный просмотр пациентов и врачей отделения

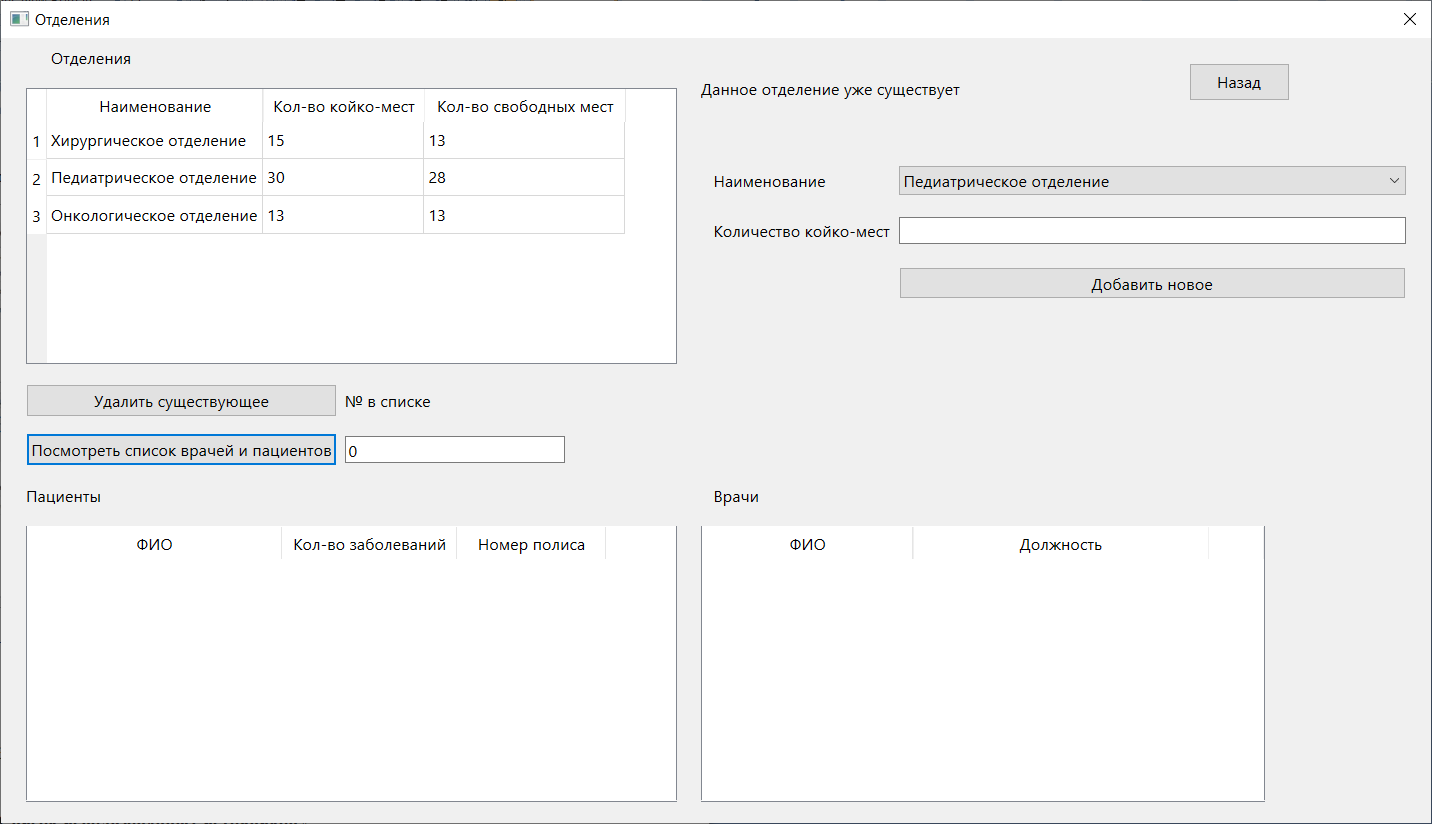


Рис. 6.19. Ввод несуществующего порядкового номера

**Удаление отделения:**

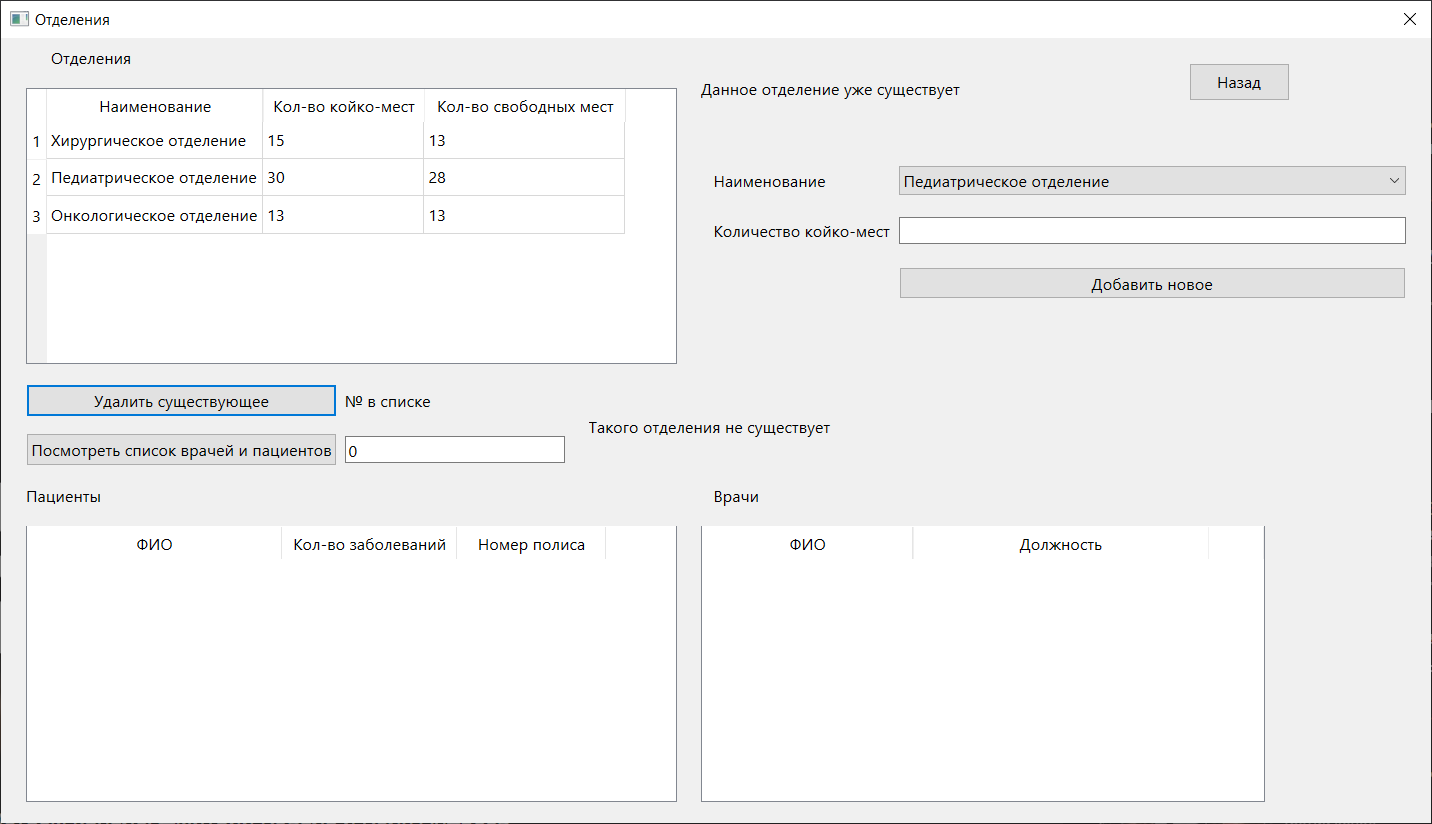
****

Рис. 6.20. Ввод несуществующего порядкового номера

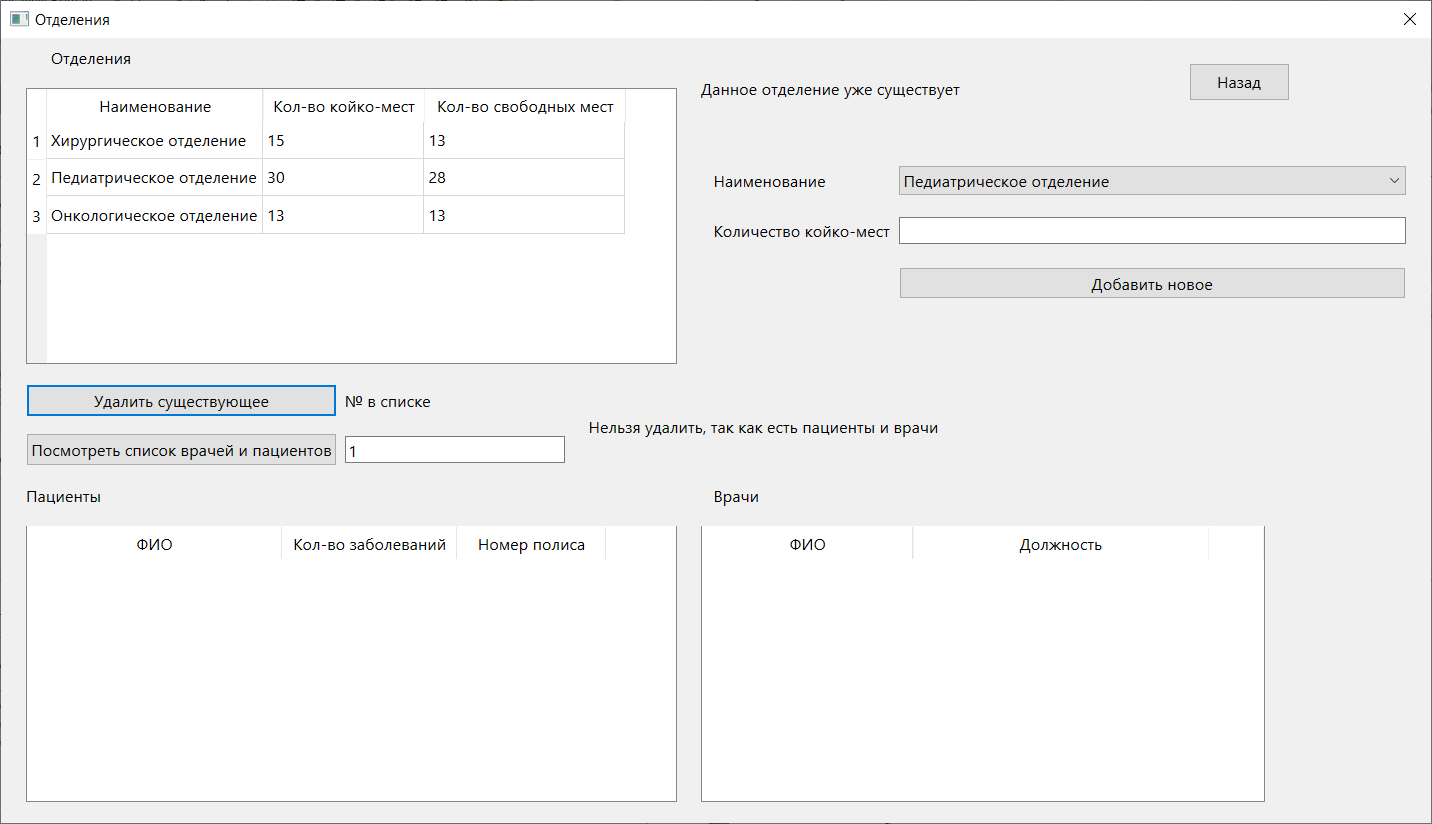


Рис. 6.21. Неверный ввод (в отделении есть пациенты и врачи)

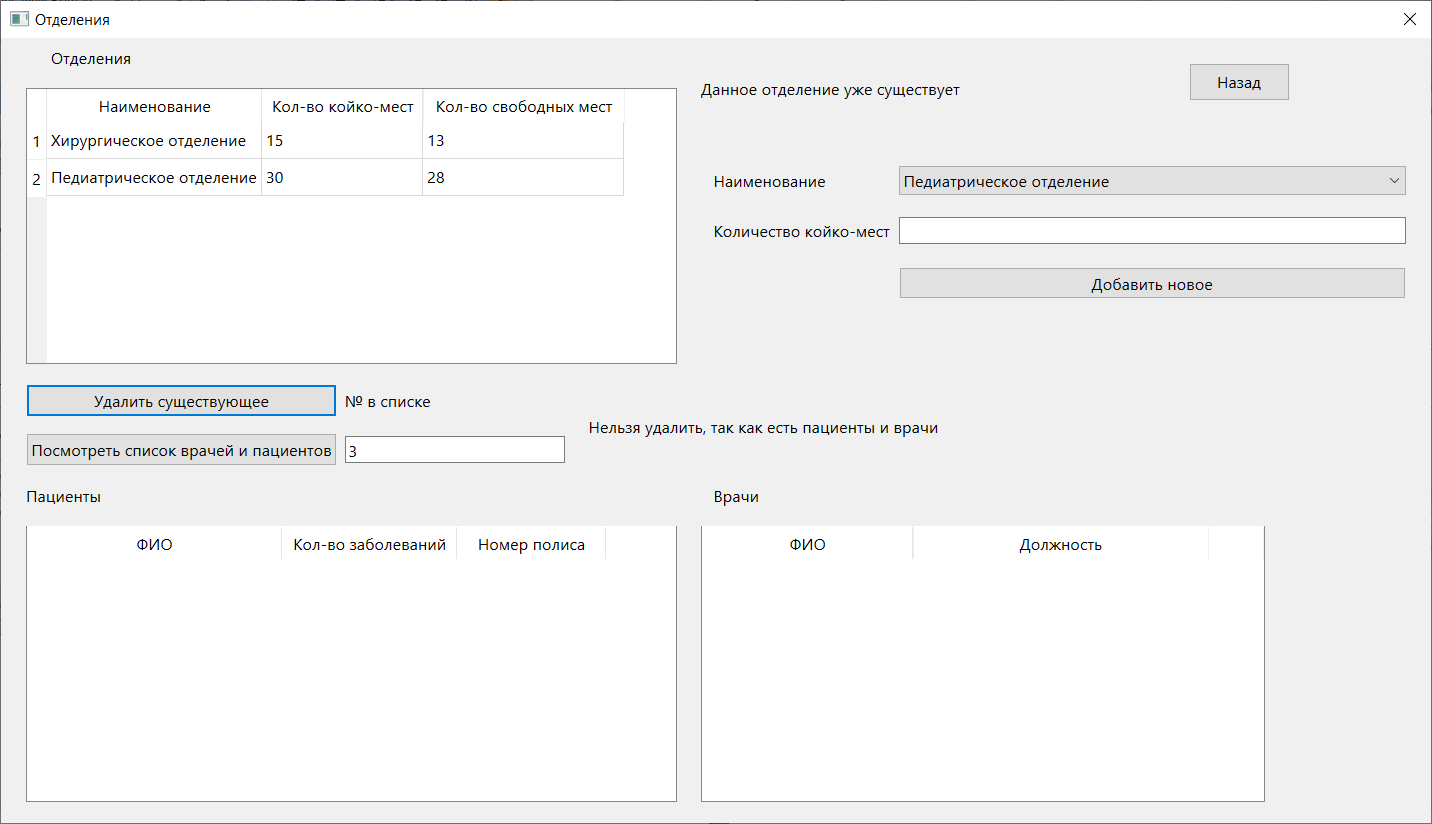


Рис. 6.22. Успешное удаление

*Диалоговое окно «Пациенты»:*

**Зарегистрировать нового пациента:**

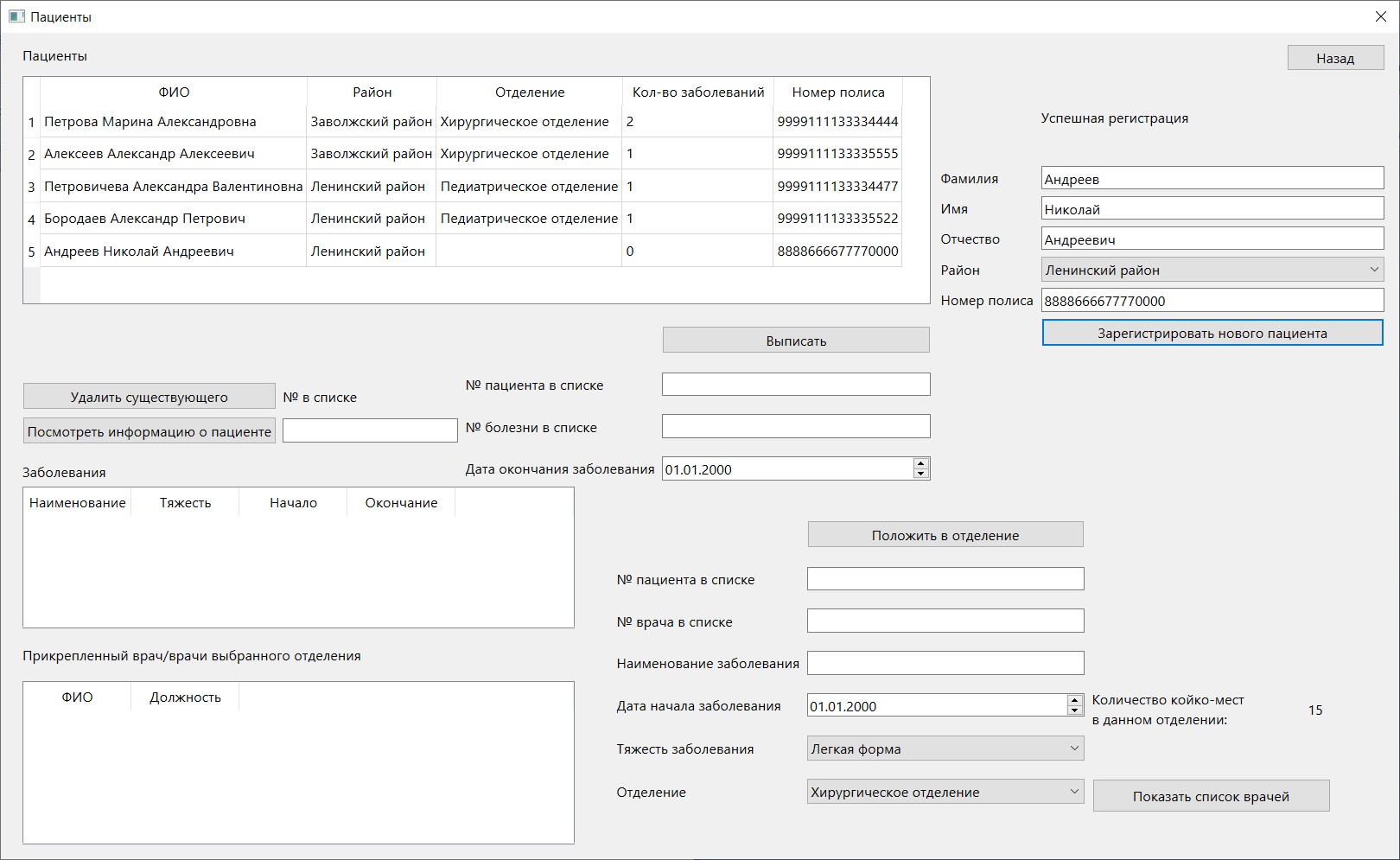


Рис. 6.23. Успешная регистрация пациента

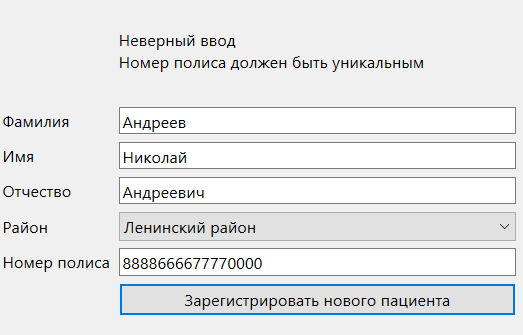


Рис. 6.24. Неверный ввод полиса

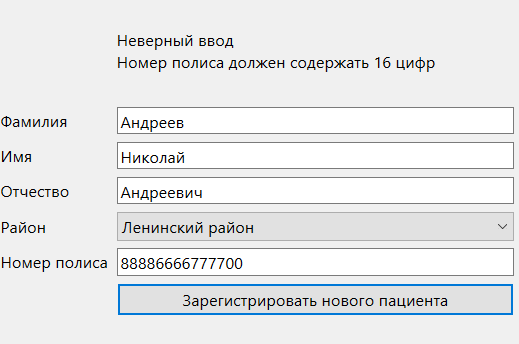


Рис. 6.25. Неверный ввод полиса (меньше 16 цифр)

**Удалить пациента:**

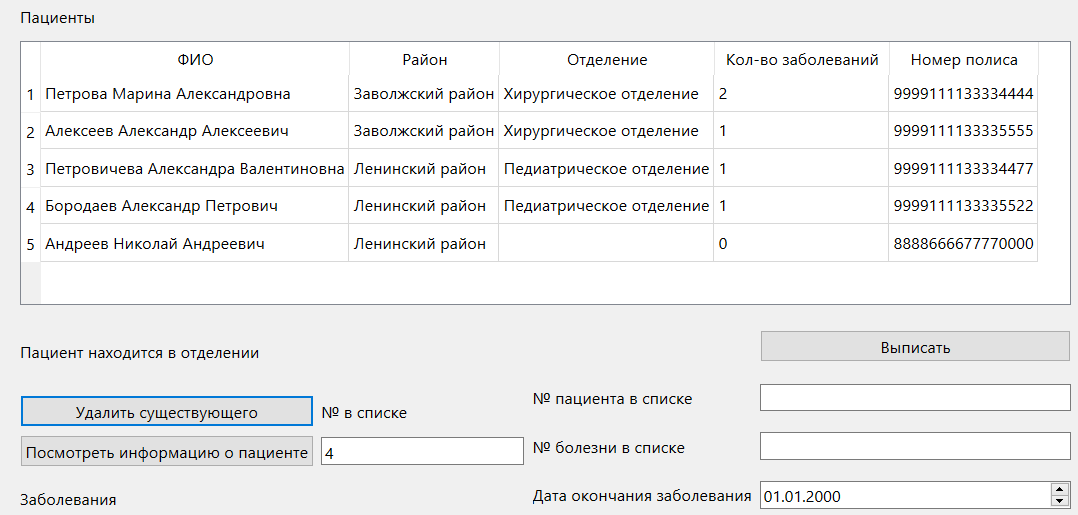


Рис. 6.25. Невозможно удалить, так как пациент в отделении

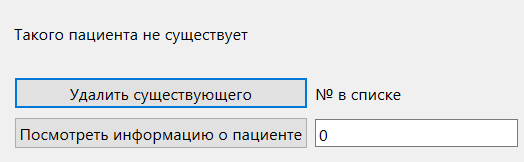


Рис. 6.26. Неверный ввод порядкового номера

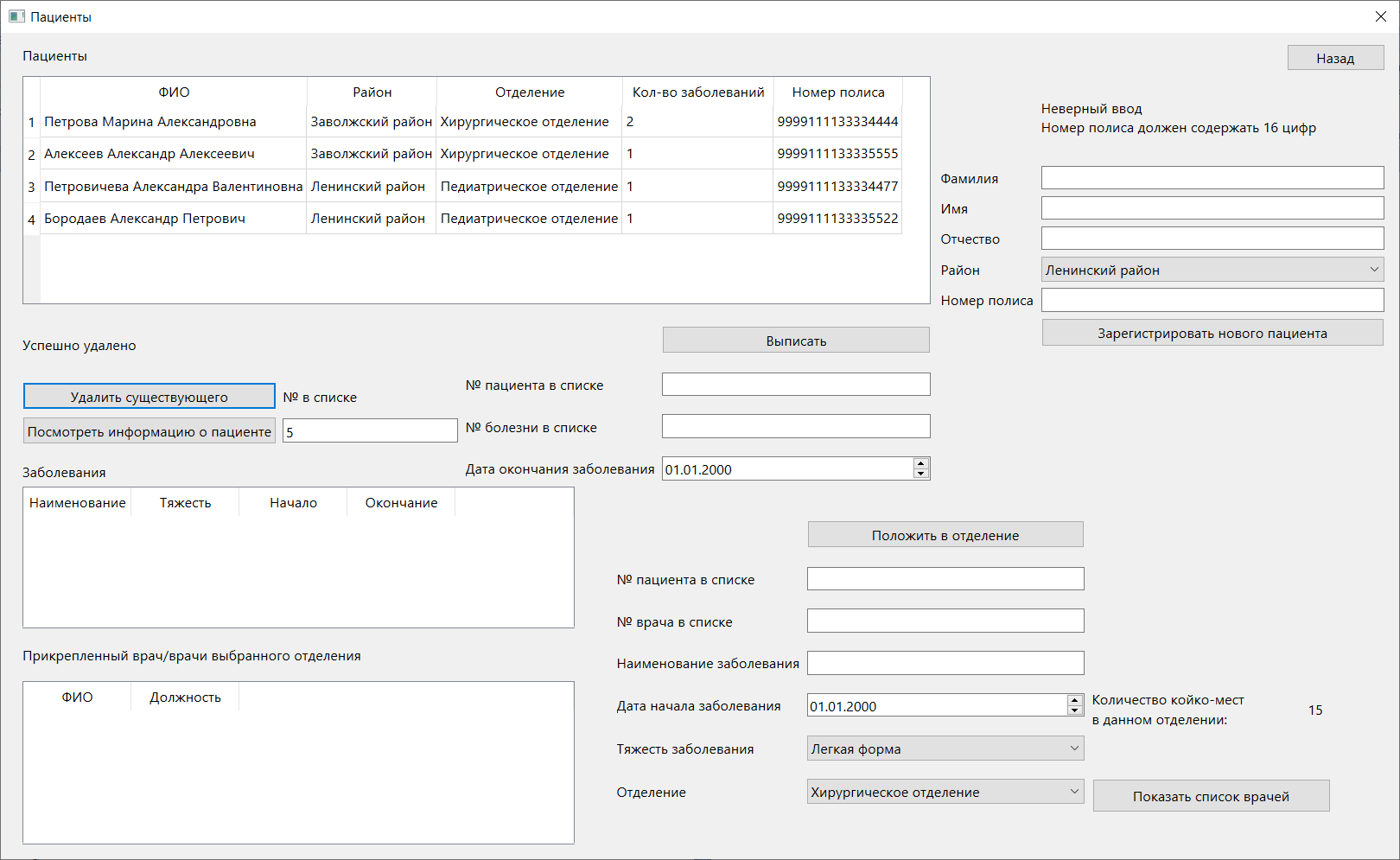


Рис. 6.27. Успешное удаление

**Выписать пациента:**

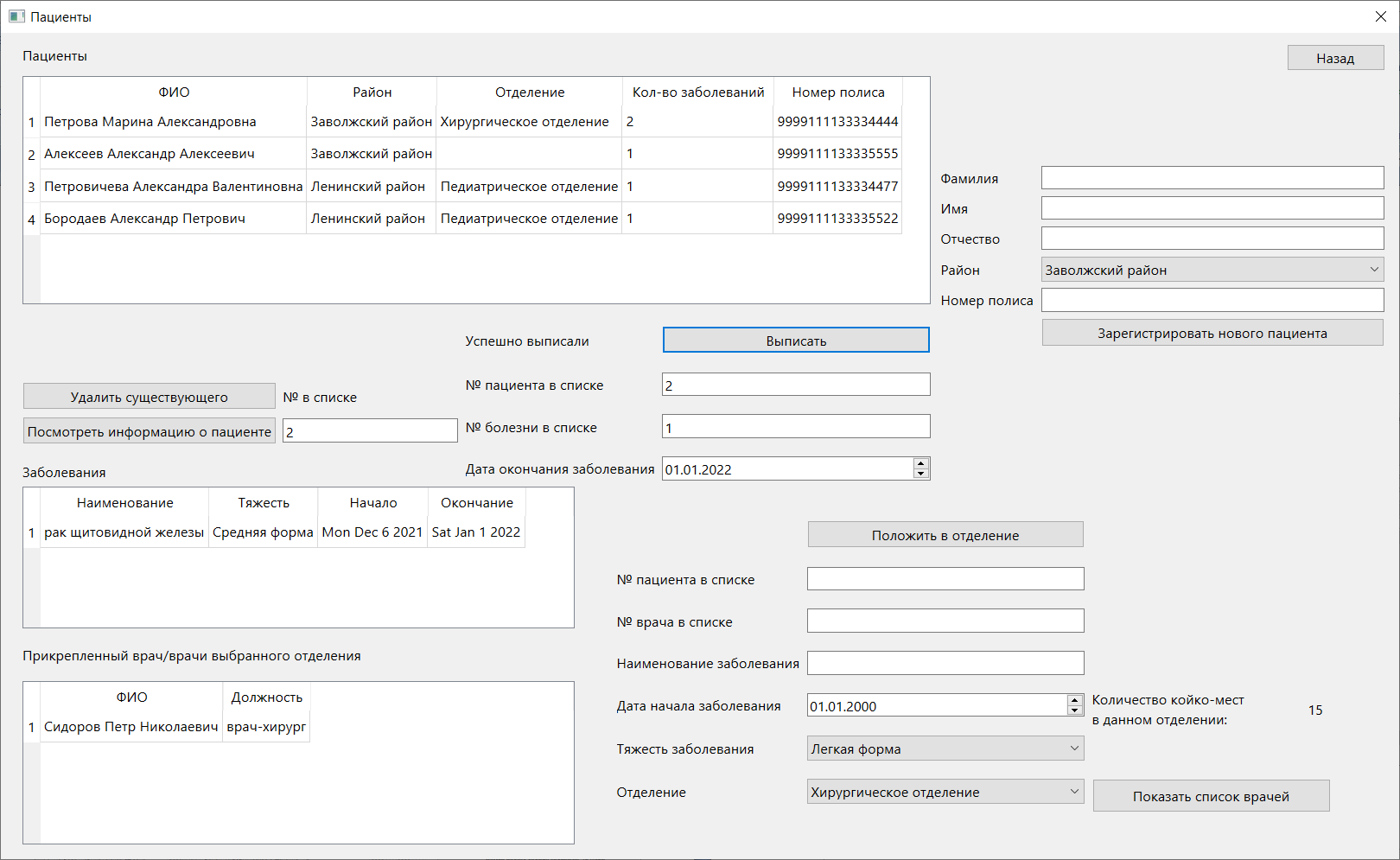


Рис. 6.28. Успешная выписка

**Положить в отделение пациента:**

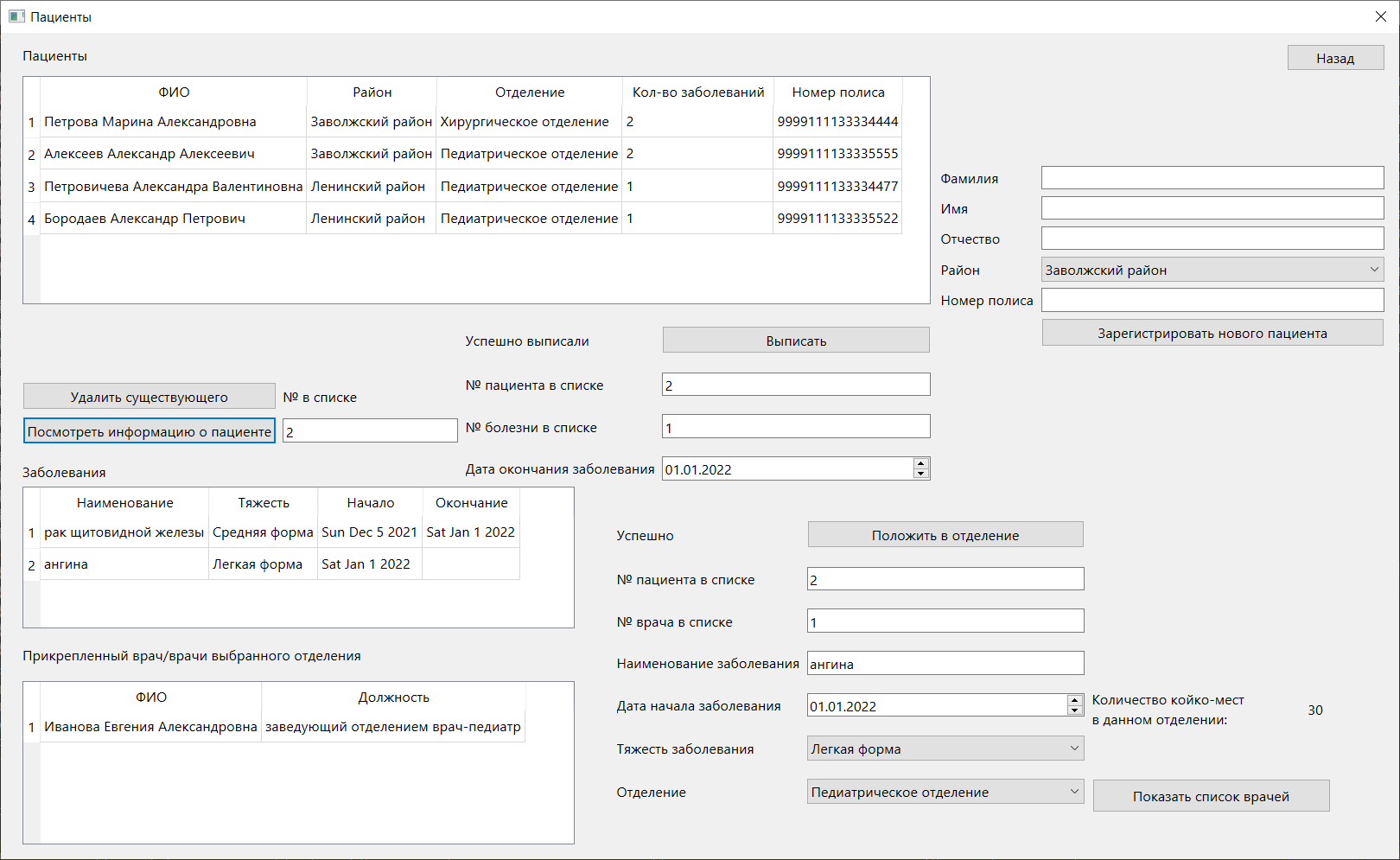


Рис. 6.29. Успешная регистрация в отделение

**Посмотреть прикрепленного врача и список болезней:**

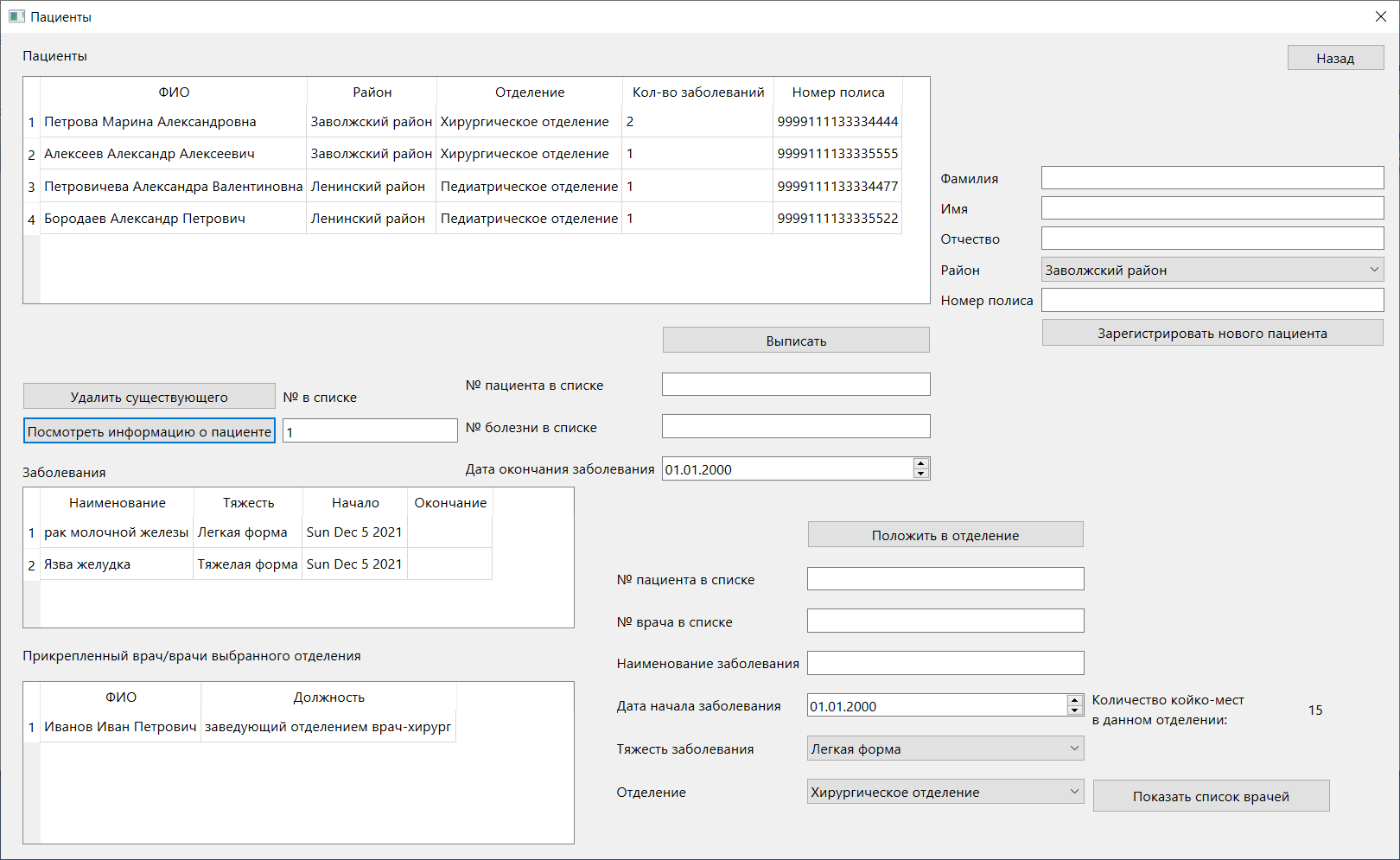


Рис. 6.30. Успешный просмотр информации о пациенте

### **Заключение**

Подводя итоги, можно выделить преимущества и недостатки разработанного приложения. К достоинствам можно отнести:

* в программе осуществляются различные проверки ввода данных;
* данная программа проста и понятна в использовании;
* многие поля ввода содержат подсказки, такие как выбор тяжести заболевания, наименования отделения;
* принимает как латиницу, так и кириллицу для ввода данных.

К недостаткам можно отнести:

* отсутствует функция редактирования для любых хранимых данных;
* не производится очистка выплывающих сообщений и полей ввода, при отправке данных с формы.

### **Список использованных источников**

1. **Шлее М., Qt 5.10. Профессиональное программирование на С++. – СПб.: БХВ-Петербург, 2018. – 1072 с.**
2. **Qt Documentation:QDialog Class [Электронный ресурс]. – The Qt Company Ltd, 2021. – URL:** <https://doc.qt.io/qt-5/qdialog.html> **(дата обращения: 05.11.2021)**
3. Е.О. Шумова, ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ Методические указания к выполнению курсового проекта. – ФГАОУ ВО СПбГУАП, 2021. – 17 с.

### **Приложение А**

**Dialog\_district.h**

#ifndef DIALOG\_DISTRICT\_H

#define DIALOG\_DISTRICT\_H

#include "hospital.h"

#include <QDialog>

*namespace* **Ui** {

*class* **Dialog\_district**;

}

*class* **Dialog\_district** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **Dialog\_district**(QWidget \*parent = *nullptr*);

~***Dialog\_district***();

*private* slots:

void **showtable**();

int **NumOfPatientInWard**(District \*d);

void **on\_back\_clicked**();

void **recieveData**(hospital \*Ho);

void **on\_delet\_clicked**();

void **on\_add\_clicked**();

void **on\_show\_2\_clicked**();

void **showtable2**(int m);

*private*:

hospital \*H;*//копия* *бд*

Ui::Dialog\_district \*ui;

};

#endif *//* *DIALOG\_DISTRICT\_H*

**Dialog\_district.cpp**

#include "dialog\_district.h"

#include "hospital.h"

#include "ui\_dialog\_district.h"

Dialog\_district::Dialog\_district(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::Dialog\_district)

{

ui->setupUi(*this*);

}

Dialog\_district::~Dialog\_district()

{

*delete* ui;

}

void Dialog\_district::on\_back\_clicked()

{

close();

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget->setRowCount(0);

ui->label\_5->clear();

ui->number->clear();

ui->name->clear();

}

void Dialog\_district::recieveData(hospital \*Ho)

{

*this*->H = Ho;*//копируем* *бд*

showtable();

ui->tableWidget\_2->resizeColumnsToContents();

ui->tableWidget->resizeColumnsToContents();

}

void Dialog\_district::showtable(){

*if*(H->hasDistrict()){

QStringList lst;

int n = H->getNumOfDistrict();

ui->tableWidget->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getDistrict(i)->getNamedistrict());

ui->tableWidget->setItem(i, j, ptwi);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(QString::number(H->getDistrict(i)->getNumOfPatient()));

ui->tableWidget->setItem(i, j, ptwi);

}

j = 2;

*for* (int i = 0; i < n; ++i) {

int m = NumOfPatientInWard(H->getDistrict(i));

ptwi = *new* QTableWidgetItem(QString::number(m));

ui->tableWidget->setItem(i, j, ptwi);

}

ui->tableWidget->resizeColumnsToContents();

}

}

void Dialog\_district::showtable2(int m){

*if*(H->getDistrict(m)->hasDpatients()){

QStringList lst;

int n = H->getDistrict(m)->getNumOfPatient();

ui->tableWidget\_2->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_2->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;int i = -1;

*foreach*(Patient\* p,H->getDistrict(m)->getDpatients()){

ptwi = *new* QTableWidgetItem(p->getSurname()

+" "+p->getName()

+" "+p->getPatronymic());

i++;

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j++;i = -1;

*foreach*(Patient\* p,H->getDistrict(m)->getDpatients()){

QString name = H->NameOfWard(p);

ptwi = *new* QTableWidgetItem(name);

i++;

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j++;i = -1;

*foreach*(Patient\* p,H->getDistrict(m)->getDpatients()){

ptwi = *new* QTableWidgetItem(QString::number(p->getNumOfDisease()));

i++;

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j++;i = -1;

*foreach*(Patient\* p,H->getDistrict(m)->getDpatients()){

ptwi = *new* QTableWidgetItem(p->getPolicy());

i++;

ui->tableWidget\_2->setItem(i, j, ptwi);

}

ui->tableWidget\_2->resizeColumnsToContents();

}

}

void Dialog\_district::on\_show\_2\_clicked()

{

ui->tableWidget\_2->setRowCount(0);

*//проверка* *на* *наличие* *n*

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n< H->getNumOfDistrict()){

showtable2(n);

}

}

int Dialog\_district::NumOfPatientInWard(District\* d){

int n=0;

*foreach*(Patient\* p,d->getDpatients())

*if*(p->getInWard())

n++;

*return* n;

}

void Dialog\_district::on\_delet\_clicked()

{

*if*(H->hasWard()){

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n< H->getNumOfDistrict()){

*if*(!H->getDistrict(n)->hasDpatients()){

H->delDistrict(n);

showtable();

}

*else*

ui->label\_7->setText("Нельзя удалить, так как есть пациенты");

}

*else*

ui->label\_7->setText("Такого района не существует");

}

}

void Dialog\_district::on\_add\_clicked()

{

ui->label\_5->clear();

QString name = ui->name->text();

*if*(H->findNameDistrict(name)){

ui->label\_5->setText("Данный район уже существует");

}

*else* *if*(!name.isEmpty()){

District\* d=*new* District;

d->setNamedistrict(name);

H->setDistrict(d);

ui->label\_5->setText("Район успешно добавлен");

showtable();

}

}

**Dialog\_doctor.h**

#ifndef DIALOG\_DOCTOR\_H

#define DIALOG\_DOCTOR\_H

#include "hospital.h"

#include <QDialog>

*namespace* **Ui** {

*class* **Dialog\_doctor**;

}

*class* **Dialog\_doctor** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **Dialog\_doctor**(QWidget \*parent = *nullptr*);

~***Dialog\_doctor***();

*private* slots:

void **on\_back\_clicked**();

void **recieveData**(hospital \*Ho);

void **showtable**();

void **showDoctorsW**(Ward \*W, int a, int n);

void **on\_add\_new\_clicked**();

void **on\_delet\_clicked**();

void **on\_show\_2\_clicked**();

void **showtablePatients**(QList<Patient\*> lst);

*private*:

int NumOfD = 0;

hospital \*H;*//копия* *бд*

Ui::Dialog\_doctor \*ui;

};

#endif *//* *DIALOG\_DOCTOR\_H*

**Dialog\_doctor.cpp**

#include "dialog\_doctor.h"

#include "hospital.h"

#include "ui\_dialog\_doctor.h"

Dialog\_doctor::**Dialog\_doctor**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::Dialog\_doctor)

{

ui->setupUi(*this*);

}

Dialog\_doctor::~***Dialog\_doctor***()

{

*delete* ui;

}

void Dialog\_doctor::**on\_back\_clicked**()

{

close();

ui->tableWidget->setRowCount(0);

ui->tableWidget\_2->setRowCount(0);

ui->number->clear();

ui->name->clear();

ui->surname->clear();

ui->patron->clear();

ui->policy->clear();

ui->position->clear();

ui->res\_add\_new->clear();

ui->label\_7->clear();

}

void Dialog\_doctor::**recieveData**(hospital \*Ho)

{

*this*->H = Ho;*//копируем* *бд*

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget->setRowCount(0);

showtable();

ui->nameward->clear();

foreach(Ward\* w, H->getListOfWard()){

ui->nameward->addItem(w->getNameward());

}

}

void Dialog\_doctor::**showDoctorsW**(Ward\* W, int a, int n){

QTableWidgetItem\* ptwi = 0;

int j=0;

*for* (int i =0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(W->getDoctor(i)->getSurname()

+" "+W->getDoctor(i)->getName()

+" "+W->getDoctor(i)->getPatronymic());

ui->tableWidget->setItem(k, j, *ptwi*);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(W->getNameward());

ui->tableWidget->setItem(k, j, *ptwi*);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(W->getDoctor(i)->getPosition());

ui->tableWidget->setItem(k, j, *ptwi*);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(W->getDoctor(i)->getPolicy());

ui->tableWidget->setItem(k, j, *ptwi*);

}

}

*//правильно* *работает*

void Dialog\_doctor::**showtable**(){

*if*(H->hasWard()){

int n (0);

foreach(Ward\* w, H->getListOfWard())

n+=w->getNumOfDoctors();

NumOfD=n;

*if*(n>0){

QStringList lst;

ui->tableWidget->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget->setVerticalHeaderLabels(lst);

bool f=0;

int i(0),m(0);

foreach(Ward\* w,H->getListOfWard()){

*if*(f){

i+=m;

}

*if*(w->hasDoctors()){

m=w->getNumOfDoctors();

showDoctorsW(*w*, i, m);

f=1;

}

}

}

*else*

ui->tableWidget->setRowCount(0);

ui->tableWidget->resizeColumnsToContents();

}

}

void Dialog\_doctor::**on\_add\_new\_clicked**()

{

QString name = ui->name->text();

QString surname = ui->surname->text();

QString pat = ui->patron->text();

QString pol = ui->policy->text();

QString pos = ui->position->text();

QString wardname = ui->nameward->currentText();

Ward\* w = H->getWardByName(wardname);

*if*(w!=*nullptr*){

*if*(pol.size()!=16){

ui->res\_add\_new->setText("Неверный ввод\nНомер полиса должен содержать 16 цифр");

}

*else* *if*(H->policyunique(pol)){

Doctor\* d = *new* Doctor();

d->setName(name);

d->setPatronymic(pat);

d->setSurname(surname);

d->setPolicy(pol);

d->setPosition(pos);

w->setDoctor(*d*);

NumOfD++;

ui->res\_add\_new->setText("Успешная регистрация");

showtable();

}

*else*{

ui->res\_add\_new->setText("Неверный ввод\nНомер полиса должен быть уникальным");

}

}

}

void Dialog\_doctor::**on\_delet\_clicked**()

{

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n<NumOfD){

QTableWidgetItem\* q = ui->tableWidget->item(n,3);

QString pol;

*if*(q!=NULL)

pol = q->text();

Doctor\* d = H->getDoctorPolicy(pol);

*if*(d!=*nullptr*){

*if*(!H->DoctorhasPatient(*d*)){*//проверка* *на* *то* *есть* *ли* *у* *него* *пациенты*

H->delDoctor(d->getPolicy());

ui->label\_7->setText("Успешно удалено");

NumOfD--;

showtable();

}

*else*

ui->label\_7->setText("У врача есть пациент(ы)");

}

}

*else*

ui->label\_7->setText("Такого врача не существует");

ui->tableWidget\_2->setRowCount(0);

}

void Dialog\_doctor::**on\_show\_2\_clicked**()

{

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n<NumOfD){

QTableWidgetItem\* q = ui->tableWidget->item(n,3);

QString pol;

*if*(q!=NULL)

pol = q->text();

Doctor\* d = H->getDoctorPolicy(pol);

*if*(d!=*nullptr*){

QList<Patient\*> list = H->DoctorhasPatients(*d*);

*if*(!list.isEmpty()){*//проверка* *на* *то* *есть* *ли* *у* *него* *пациенты*

showtablePatients(list);

}

*else*

ui->label\_7->setText("У врача нет пациентов");

}

}

*else*

ui->label\_7->setText("Такого врача не существует");

}

void Dialog\_doctor::**showtablePatients**(QList<Patient\*> list){

QStringList lst;

int n = list.count();

ui->tableWidget\_2->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_2->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(list.at(i)->getSurname()

+" "+list.at(i)->getName()

+" "+list.at(i)->getPatronymic());

ui->tableWidget\_2->setItem(i, j, *ptwi*);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(QString::number(list.at(i)->getNumOfDisease()));

ui->tableWidget\_2->setItem(i, j, *ptwi*);

}

ui->tableWidget\_2->resizeColumnsToContents();

}

**Dialog\_patient.h**

#ifndef DIALOG\_PATIENT\_H

#define DIALOG\_PATIENT\_H

#include <QDialog>

#include "hospital.h"

*namespace* **Ui** {

*class* **Dialog\_patient**;

}

*class* **Dialog\_patient** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **Dialog\_patient**(QWidget \*parent = *nullptr*);

~***Dialog\_patient***();

*private* slots:

void **on\_back\_clicked**();

void **recieveData**(hospital \*H);*//получить* *данные* *и* *скопировать*

void **showtable**();

void **showPatientD**(District \*d, int a, int n);

void **on\_show\_2\_clicked**();

void **showtable3**(Patient\* p);

void **showtable2**(Patient\* p);

void **on\_delet\_clicked**();

void **on\_writeout\_clicked**();

void **on\_add\_new\_clicked**();

void **on\_show\_doc\_clicked**();

void **on\_add\_to\_ward\_clicked**();

void **on\_nameward\_currentTextChanged**(*const* QString &arg1);

*private*:

hospital \*H;*//копия* *бд*

int NumOfP;

QList<Patient\*> ListOfP;

Ui::Dialog\_patient \*ui;

};

#endif *//* *DIALOG\_PATIENT\_H*

**Dialog\_patient.cpp**

#include "dialog\_patient.h"

#include "ui\_dialog\_patient.h"

Dialog\_patient::Dialog\_patient(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::Dialog\_patient)

{

ui->setupUi(*this*);

}

Dialog\_patient::~Dialog\_patient()

{

*delete* ui;

}

void Dialog\_patient::on\_back\_clicked()

{

close();*//закрываем* *окно*

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

ui->number->clear();

}

void Dialog\_patient::showPatientD(District\* D, int a, int n){

QTableWidgetItem\* ptwi = 0;

int j=0;

*for* (int i =0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(D->getDpatient(i)->getSurname()

+" "+D->getDpatient(i)->getName()

+" "+D->getDpatient(i)->getPatronymic());

ui->tableWidget->setItem(k, j, ptwi);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(D->getNamedistrict());

ui->tableWidget->setItem(k, j, ptwi);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

QString name = H->NameOfWard(D->getDpatient(i));

ptwi = *new* QTableWidgetItem(name);

ui->tableWidget->setItem(k, j, ptwi);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(QString::number(D->getDpatient(i)->getNumOfDisease()));

ui->tableWidget->setItem(k, j, ptwi);

}

j++;

*for* (int i = 0, k=a; i < n; ++i,++k) {

ptwi = *new* QTableWidgetItem(D->getDpatient(i)->getPolicy());

ui->tableWidget->setItem(k, j, ptwi);

}

}

void Dialog\_patient::showtable(){

*if*(H->hasDistrict()){

int n (0);

*foreach*(District\* d, H->getListOfDistrict())

n+=d->getNumOfPatient();

NumOfP=n;

*if*(n>0){

QStringList lst;

ui->tableWidget->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget->setVerticalHeaderLabels(lst);

bool f=0;

int i(0),m(0);

*foreach*(District\* d,H->getListOfDistrict()){

*if*(f){

i+=m;

}

*if*(d->hasDpatients()){

m=d->getNumOfPatient();

showPatientD(d, i, m);

f=1;

}

}

}

*else*

ui->tableWidget->setRowCount(0);

ui->tableWidget->resizeColumnsToContents();

}

}

void Dialog\_patient::on\_show\_2\_clicked()

{

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n<NumOfP){

QTableWidgetItem\* q = ui->tableWidget->item(n,4);

QString pol;

*if*(q!=NULL)

pol = q->text();

Patient\* p = H->getPatientPolicy(pol);

*if*(p!=*nullptr*){

showtable2(p);

showtable3(p);

}

}

*else*{

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

}

}

void Dialog\_patient::showtable2( Patient\* p){

*if*(p->getDoc()!=NULL){

QStringList lst;

ui->tableWidget\_2->setRowCount(1);

lst<< "1";

ui->tableWidget\_2->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*//ФИО* *врача*

ptwi = *new* QTableWidgetItem(p->getDoc()->getSurname()

+" "+p->getDoc()->getName()

+" "+p->getDoc()->getPatronymic());

ui->tableWidget\_2->setItem(0, j, ptwi);

*//должность*

j = 1;

ptwi = *new* QTableWidgetItem(p->getDoc()->getPosition());

ui->tableWidget\_2->setItem(0, j, ptwi);

ui->tableWidget\_2->resizeColumnsToContents();

}

}

void Dialog\_patient::showtable3( Patient\* p){

*if*(p->hasDiseases()){

QStringList lst;

int n = p->getPdiseases().count();

ui->tableWidget\_3->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_3->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*//наименование* *болезни*

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(p->getPdiseas(i)->getNameOfdisease());

ui->tableWidget\_3->setItem(i, j, ptwi);

}

*//сложность*

j = 1;

*for* (int i = 0; i < n; ++i) {

Disease::severity s = p->getPdiseas(i)->getSev();

*if*(s==Disease::severity::light)

ptwi = *new* QTableWidgetItem("Легкая форма");

*else* *if*(s==Disease::severity::medium)

ptwi = *new* QTableWidgetItem("Средняя форма");

*else* *if*(s==Disease::severity::heavy)

ptwi = *new* QTableWidgetItem("Тяжелая форма");

ui->tableWidget\_3->setItem(i, j, ptwi);

}

*//дата* *начала* *болезни*

j = 2;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(p->getPdiseas(i)->getStart().toString());

ui->tableWidget\_3->setItem(i, j, ptwi);

}

*//дата* *окончания*

j = 3;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(p->getPdiseas(i)->getFinish().toString());

ui->tableWidget\_3->setItem(i, j, ptwi);

}

ui->tableWidget\_3->resizeColumnsToContents();

}

}

void Dialog\_patient::recieveData(hospital \*Ho)

{

*this*->H = Ho;*//копируем* *бд*

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

ui->district->clear();

*foreach*(District\* d,H->getListOfDistrict()){

ui->district->addItem(d->getNamedistrict());

}

ui->nameward->clear();

*foreach*(Ward\* w, H->getListOfWard()){

ui->nameward->addItem(w->getNameward());

}

showtable();

}

void Dialog\_patient::on\_delet\_clicked()

{

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n<NumOfP){

QTableWidgetItem\* q = ui->tableWidget->item(n,4);

QString pol;

*if*(q!=NULL)

pol = q->text();

Patient\* p = H->getPatientPolicy(pol);

*if*(p!=*nullptr*){

*if*(!p->getInWard()){

H->delPatient(p->getId());

ui->label\_7->setText("Успешно удалено");

NumOfP--;

showtable();

}

*else*

ui->label\_7->setText("Пациент находится в отделении");

}

}

*else*

ui->label\_7->setText("Такого пациента не существует");

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

}

void Dialog\_patient::on\_writeout\_clicked()

{

int n = ui->number\_p\_d->text().toInt();

n--;

int m = ui->number\_pd\_2->text().toInt();

m--;

QDate d = ui->date\_end->date();

*if*(n>=0 && n<NumOfP){

QTableWidgetItem\* q = ui->tableWidget->item(n,4);

QString pol;

*if*(q!=NULL)

pol = q->text();

Patient\* p = H->getPatientPolicy(pol);

*if*(p!=*nullptr*)

*if*(p->hasDiseases()){

*if*(m>=0 && m<p->getNumOfDisease())

*if*(d>p->getPdiseas(m)->getStart()){

p->getPdiseas(m)->setFinish(d);

ui->res\_add\_date->setText("Успешно выписали");

showtable3(p);

showtable();

}

*foreach*(Disease\* dis, p->getPdiseases())

{

*if*(!dis->getFinish().isNull()){

p->setInWard(0);

H->delPatientfromWard(p->getId());

}

}

}

}

}

void Dialog\_patient::on\_add\_new\_clicked()

{

QString name = ui->name->text();

QString surname = ui->surname->text();

QString pat = ui->patron->text();

QString pol = ui->policy->text();

QString disname = ui->district->currentText();

*if*(pol.size()!=16){

ui->res\_add\_new->setText("Неверный ввод\nНомер полиса должен содержать 16 цифр");

}

*else* *if*(H->policyunique(pol)){

Patient\* p = *new* Patient();

p->setId(H->getCurrentId());

H->setCurrentId(H->getCurrentId()+1);

p->setName(name);

p->setPatronymic(pat);

p->setSurname(surname);

p->setPolicy(pol);

H->getDistrictAndPut(disname,p);

NumOfP++;

ui->res\_add\_new->setText("Успешная регистрация");

showtable();

}

*else*{

ui->res\_add\_new->setText("Неверный ввод\nНомер полиса должен быть уникальным");

}

}

void Dialog\_patient::on\_show\_doc\_clicked()

{

Ward \*w = H->getWardByName(ui->nameward->currentText());

*if*(w!=*nullptr*){

*if*(w->hasDoctors()){

QStringList lst;

int n = w->getNumOfDoctors();

ui->tableWidget\_2->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_2->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(w->getDoctor(i)->getSurname()

+" "+w->getDoctor(i)->getName()

+" "+w->getDoctor(i)->getPatronymic());

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(w->getDoctor(i)->getPosition());

ui->tableWidget\_2->setItem(i, j, ptwi);

}

ui->tableWidget\_2->resizeColumnsToContents();

}

}

}

void Dialog\_patient::on\_add\_to\_ward\_clicked()

{

Ward \*w = H->getWardByName(ui->nameward->currentText());

*if*(w!=*nullptr*){

int np = ui->number\_p->text().toInt();

np--;

int nd = ui->number\_doc->text().toInt();

nd--;

QString namedis = ui->namedisea->text();

QDate dst = ui->date\_start->date();

*if*(np>=0 && np<NumOfP && nd>=0 && nd< w->getNumOfDoctors()

&& w->getNumOfPatient()<w->getNumOfBed() && !namedis.isEmpty())

{

QTableWidgetItem\* q = ui->tableWidget->item(np,4);

QString pol;

*if*(q!=NULL)

pol = q->text();

Patient\* p = H->getPatientPolicy(pol);

*if*(!p->getInWard()){

Disease\* d = *new* Disease;

d->setNameOfdisease(namedis);

d->setSev(ui->severity->currentIndex());

d->setStart(dst);

p->setPdiseas(d);

p->setDoc(w->getDoctor(nd));

p->setInWard(1);

w->setWpatient(p);

ui->res\_add\_w->setText("Успешно");

showtable();

}

*else*

ui->res\_add\_w->setText("Пациент уже находится\nв отделении больницы");

}

*else*

ui->res\_add\_w->setText("Неверный ввод");

}

}

void Dialog\_patient::on\_nameward\_currentTextChanged(*const* QString &arg1)

{

Ward \*w = H->getWardByName(arg1);

*if*(w!=*nullptr*)

ui->numOfbed->setText(QString::number(w->getNumOfBed()));

}

**Dialog\_ward.h**

#ifndef DIALOG\_WARD\_H

#define DIALOG\_WARD\_H

#include <QDialog>

#include "hospital.h"

*namespace* **Ui** {

*class* **Dialog\_ward**;

}

*class* **Dialog\_ward** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **Dialog\_ward**(QWidget \*parent = *nullptr*);

~***Dialog\_ward***();

*private* slots:

void **showtable**();

void **showtable2**(int n);

void **showtable3**(int m);

void **on\_back\_clicked**();*//закрыть* *окно*

void **recieveData**(hospital \*H);*//получить* *данные* *и* *скопировать*

void **on\_show\_2\_clicked**();

void **on\_delet\_clicked**();

void **on\_add\_clicked**();

*private*:

hospital \*H;*//копия* *бд*

Ui::Dialog\_ward \*ui;

};

#endif *//* *DIALOG\_WARD\_H*

**Dialog\_ward.cpp**

#include "dialog\_ward.h"

#include "ui\_dialog\_ward.h"

Dialog\_ward::Dialog\_ward(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::Dialog\_ward)

{

ui->setupUi(*this*);

}

Dialog\_ward::~Dialog\_ward()

{

*delete* ui;

}

void Dialog\_ward::on\_back\_clicked()

{

close();*//закрываем* *окно*

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

ui->label\_5->clear();

ui->number->clear();

ui->number\_place->clear();

}

void Dialog\_ward::showtable(){

*if*(H->hasWard()){

QStringList lst;

int n = H->getNumOfWard();

ui->tableWidget->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getWard(i)->getNameward());

ui->tableWidget->setItem(i, j, ptwi);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(QString::number(H->getWard(i)->getNumOfBed()));

ui->tableWidget->setItem(i, j, ptwi);

}

j = 2;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(QString::number(H->getWard(i)->getNumOfBed()-H->getWard(i)->getNumOfPatient()));

ui->tableWidget->setItem(i, j, ptwi);

}

ui->tableWidget->resizeColumnsToContents();

}

}

void Dialog\_ward::recieveData(hospital \*Ho)

{

*this*->H = Ho;*//копируем* *бд*

showtable();

ui->tableWidget\_2->resizeColumnsToContents();

ui->tableWidget\_3->resizeColumnsToContents();

}

void Dialog\_ward::showtable2(int m){

*if*(H->getWard(m)->hasPatients()){

QStringList lst;

int n = H->getWard(m)->getNumOfPatient();

ui->tableWidget\_2->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_2->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getWard(m)->getWpatient(i)->getSurname()

+" "+H->getWard(m)->getWpatient(i)->getName()

+" "+H->getWard(m)->getWpatient(i)->getPatronymic());

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(QString::number(H->getWard(m)->getWpatient(i)->getNumOfDisease()));

ui->tableWidget\_2->setItem(i, j, ptwi);

}

j = 2;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getWard(m)->getWpatient(i)->getPolicy());

ui->tableWidget\_2->setItem(i, j, ptwi);

}

ui->tableWidget\_2->resizeColumnsToContents();

}

}

void Dialog\_ward::showtable3(int m){

*if*(H->getWard(m)->hasDoctors()){

QStringList lst;

int n = H->getWard(m)->getNumOfDoctors();

ui->tableWidget\_3->setRowCount(n);

*for*(int i=1;i<=n;i++)

lst<< QString::number(i);

ui->tableWidget\_3->setVerticalHeaderLabels(lst);

QTableWidgetItem\* ptwi = 0;

int j = 0;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getWard(m)->getDoctor(i)->getSurname()

+" "+H->getWard(m)->getDoctor(i)->getName()

+" "+H->getWard(m)->getDoctor(i)->getPatronymic());

ui->tableWidget\_3->setItem(i, j, ptwi);

}

j = 1;

*for* (int i = 0; i < n; ++i) {

ptwi = *new* QTableWidgetItem(H->getWard(m)->getDoctor(i)->getPosition());

ui->tableWidget\_3->setItem(i, j, ptwi);

}

ui->tableWidget\_3->resizeColumnsToContents();

}

}

void Dialog\_ward::on\_show\_2\_clicked()

{

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_3->setRowCount(0);

*//проверка* *на* *наличие* *n*

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n< H->getNumOfWard()){

showtable2(n);

showtable3(n);

}

}

void Dialog\_ward::on\_delet\_clicked()

{

*if*(H->hasWard()){

int n = ui->number->text().toInt();

n--;

*if*(n>=0 && n< H->getNumOfWard()){

*if*(!H->getWard(n)->hasPatients() && !H->getWard(n)->hasDoctors()){

H->delWard(n);

showtable();

}

*else*

ui->label\_5->setText("Нельзя удалить, так как есть пациенты и врачи");

}

*else*

ui->label\_5->setText("Такого отделения не существует");

}

}

void Dialog\_ward::on\_add\_clicked()

{

ui->label\_5->clear();

int n = ui->number\_place->text().toInt();

QString name = ui->comboBox\_name->currentText();

*if*(H->findNameWard(name)){

ui->label\_8->setText("Данное отделение уже существует");

}

*else* *if*(n>0){

Ward\* w=*new* Ward;

w->setNameward(name);

w->setNumOfBed(n);

H->setWard(w);

ui->label\_8->setText("Отделение успешно добавлено");

showtable();

}

*else*{

ui->label\_8->setText("Неверный ввод");

}

}

**Disease.h**

#ifndef DISEASE\_H

#define DISEASE\_H

#include"QString"

#include"QDate"

*class* **Disease**

{

*public*:

*//степень* *тяжести* *болезни*

*enum* **severity**{

*light*,

*medium*,

*heavy*

};

**Disease**();

*//сеттеры* *и* *геттеры* *для* *хранимых* *данных*

severity **getSev**() *const*;

void **setSev**(int i);

*const* QString &**getNameOfdisease**() *const*;

void **setNameOfdisease**(*const* QString &newNameOfdisease);

*const* QDate &**getStart**() *const*;

void **setStart**(*const* QDate &newStart);

*const* QDate &**getFinish**() *const*;

void **setFinish**(*const* QDate &newFinish);

*private*:

severity Sev; *//тяжесть* *болезни*

QString nameOfdisease; *//название*

QDate start; *//дата* *начала* *болезни*

QDate finish; *//дата* *окончания*

};

#endif *//* *DISEASE\_H*

**Disease.cpp**

#include "disease.h"

Disease::**Disease**()

{

}

Disease::severity Disease::**getSev**() *const*

{

*return* Sev;

}

void Disease::**setSev**(int i)

{

*if*(i==0)

Sev = severity::*light*;

*else* *if*(i==1)

Sev = severity::*medium*;

*else* *if*(i==2)

Sev = severity::*heavy*;

}

*const* QString &Disease::**getNameOfdisease**() *const*

{

*return* nameOfdisease;

}

void Disease::**setNameOfdisease**(*const* QString &newNameOfdisease)

{

nameOfdisease = newNameOfdisease;

}

*const* QDate &Disease::**getStart**() *const*

{

*return* start;

}

void Disease::**setStart**(*const* QDate &newStart)

{

start = newStart;

}

*const* QDate &Disease::**getFinish**() *const*

{

*return* finish;

}

void Disease::**setFinish**(*const* QDate &newFinish)

{

finish = newFinish;

}

**District.h**

#ifndef DISTRICT\_H

#define DISTRICT\_H

#include "patient.h"

#include "QString"

#include "QList"

*class* **District**

{

*public*:

**District**();

~**District**();

*//сеттеры* *и* *геттеры* *для* *хранимых* *данных*

*const* QString &**getNamedistrict**() *const*;

void **setNamedistrict**(*const* QString &newNamedistrict);

Patient \***getDpatient**(int n);

void **setDpatient**(Patient \*&newDpatient);

*const* QList<Patient \*> &**getDpatients**() *const*;

*//проверка* *есть* *ли* *пациент* *в* *списке* *по* *Id*

bool **hasDpatientId**(int n);

*//получение* *числа* *пациентов* *в* *списке*

int **getNumOfPatient**();

*//проверка* *есть* *ли* *пациенты* *в* *списке*

bool **hasDpatients**();

*//удаление* *пациента* *по* *Id*

void **delDpatientId**(int id);

*private*:

QString namedistrict; *//название* *района*

QList<Patient\*> dpatients; *//пациенты*

*//в* *список* *пациентов* *будут* *заноситься/регистрироваться* *пациенты,* *которые* *могут* *не* *лежать* *в* *отделении*

};

#endif *//* *DISTRICT\_H*

**District.cpp**

#include "district.h"

District::District()

{

}

District::~District()

{

qDeleteAll(dpatients);

}

bool District::hasDpatients(){

*if*(dpatients.empty())

*return* *false*;

*else*

*return* *true*;

}

Patient \* District::getDpatient(int n)

{

*return* dpatients.at(n);

}

void District::setDpatient(Patient \* &newDpatient)

{

dpatients.append(newDpatient);

}

*const* QString &District::getNamedistrict() *const*

{

*return* namedistrict;

}

void District::setNamedistrict(*const* QString &newNamedistrict)

{

namedistrict = newNamedistrict;

}

bool District::hasDpatientId(int n)

{

*foreach*(Patient\* p, dpatients)

{

*if*(p->getId()==n)

*return* *true*;

}

*return* *false*;

}

void District::delDpatientId(int id)

{

int i(0);

*foreach*(Patient\* p, dpatients){

*if*(p->getId()==id)

dpatients.takeAt(i);

i++;

}

}

int District::getNumOfPatient(){

*return* dpatients.count();

}

*const* QList<Patient \*> &District::getDpatients() *const*

{

*return* dpatients;

}

**Doctor.h**

#ifndef DOCTOR\_H

#define DOCTOR\_H

#include "human.h"

#include "QList"

#include "QString"

*class* **Doctor** : *public* Human

{

*public*:

**Doctor**();

*//сеттер* *и* *геттер* *для* *хранимого* *данного*

*const* QString &**getPosition**() *const*;

void **setPosition**(*const* QString &newPosition);

*private*:

QString position; *//должность*

};

#endif *//* *DOCTOR\_H*

**Doctor.cpp**

#include "doctor.h"

Doctor::**Doctor**()

{

}

*const* QString &Doctor::**getPosition**() *const*

{

*return* position;

}

void Doctor::**setPosition**(*const* QString &newPosition)

{

position = newPosition;

}

**hospital.h**

#ifndef HOSPITAL\_H

#define HOSPITAL\_H

#include "ward.h"

#include "district.h"

#include "QList"

*class* **hospital**

{

*public*:

*static* hospital\* **gethospital**();

*//проверка* *есть* *ли* *отделения*

bool **hasWard**();

*//проверка* *есть* *ли* *районы*

bool **hasDistrict**();

~**hospital**();

*//геттеры* *и* *сеттеры* *для* *хранимых* *данных*

*const* QString &**getAddress**() *const*;

void **setAddress**(*const* QString &newAddress);

*const* QString &**getName**() *const*;

void **setName**(*const* QString &newName);

Ward \***getWard**(int n) ;

*const* QList<Ward \*> &**getListOfWard**() *const*;

void **setWard**(Ward \*&newWard);

District \* **getDistrict**(int n) ;

*const* QList<District \*> &**getListOfDistrict**() *const*;

void **setDistrict**(District \* &newListOfDistrict);

int **getCurrentId**() *const*;

void **setCurrentId**(int newCurrentId);

int **getNumOfWard**();

*//удаление* *отделения* *по* *индексу*

void **delWard**(int n);

*//проверка* *есть* *ли* *отделение* *с* *таким* *именем*

bool **findNameWard**(QString name);

*//получение* *количества* *районов*

int **getNumOfDistrict**();

*//удаление* *района* *по* *индексу*

void **delDistrict**(int n);

*//проверка* *есть* *ли* *район* *с* *таким* *именем*

bool **findNameDistrict**(QString name);

*//получение* *названия* *отделения* *по* *пациенту*

QString **NameOfWard**(Patient\* p);

*//удаление* *пациента* *по* *Id*

void **delPatient**(int id);

*//проверка* *является* *ли* *полис* *уникальным* *среди* *людей*

bool **policyunique**(QString pol);

*//получение* *названия* *района* *по* *пациенту*

QString **NameOfDistrict**(Patient \*p);

*//получение* *пациента* *по* *полису*

Patient \***getPatientPolicy**(QString pol);

*//добавление* *пациента* *в* *район* *по* *названию* *района*

void **getDistrictAndPut**(QString str, Patient \*p);

*//получение* *отделения* *по* *названию*

Ward \***getWardByName**(QString strname);

*//получение* *врача* *по* *полису*

Doctor \***getDoctorPolicy**(QString pol);

*//удаление* *врача* *по* *полису*

void **delDoctor**(QString pol);

*//проверка* *есть* *ли* *у* *доктора* *пациенты*

bool **DoctorhasPatient**(Doctor \*d);

*//получение* *списка* *пациентов* *по* *доктору*

QList<Patient\*> **DoctorhasPatients**(Doctor \*d);

*//удаление* *пациента* *из* *списка* *отделения*

void **delPatientfromWard**(int id);

*private*:

**hospital**();

*static* hospital\* Hospital;

int currentId;

QString address; *//адрес* *больницы*

QString name; *//название* *больницы*

QList <Ward\*> ListOfWard; *//список* *с* *отделениями*

QList <District\*> ListOfDistrict;*//список* *с* *районами*

};

#endif *//* *HOSPITAL\_H*

**hospital.cpp**

#include "hospital.h"

#include "ward.h"

#include "district.h"

#include "disease.h"

hospital\* hospital::Hospital = 0;

hospital\* hospital::**gethospital**()

{

*if*(Hospital==*nullptr*){

Hospital = *new* hospital();

}

*return* Hospital;

}

*//проверка* *на* *непустой* *список*

bool hospital::**hasWard**()

{

*if*(ListOfWard.empty())

*return* *false*;

*else*

*return* *true*;

}

*//проверка* *на* *непустой* *список*

bool hospital::**hasDistrict**()

{

*if*(ListOfDistrict.empty())

*return* *false*;

*else*

*return* *true*;

}

bool hospital::**findNameWard**(QString name){

foreach(Ward\* w, ListOfWard){

*if*(name==w->getNameward())

*return* *true*;

}

*return* *false*;

}

bool hospital::**findNameDistrict**(QString name){

foreach(District\* d, ListOfDistrict){

*if*(name==d->getNamedistrict())

*return* *true*;

}

*return* *false*;

}

QString hospital::**NameOfWard**(Patient\* p){

foreach(Ward\* w, *this*->getListOfWard()){

foreach(Patient\* p2,w->getWpatients()){

*if*(p->getId()==p2->getId())

*return* w->getNameward();

}

}

*return* *nullptr*;

}

QString hospital::**NameOfDistrict**(Patient\* p){

foreach(District\* d, ListOfDistrict){

foreach(Patient\* p2,d->getDpatients()){

*if*(p->getId()==p2->getId())

*return* d->getNamedistrict();

}

}

*return* *nullptr*;

}

bool hospital::**policyunique**(QString pol){

foreach(District\* d, ListOfDistrict){

foreach(Patient\* p,d->getDpatients())

*if*(p->getPolicy()==pol)

*return* *false*;

}

foreach(Ward\* w, ListOfWard){

foreach(Patient\* p,w->getWpatients())

*if*(p->getPolicy()==pol )

*return* *false*;

foreach(Doctor\* d,w->getDoctors())

*if*(d->getPolicy()==pol )

*return* *false*;

}

*return* *true*;

}

hospital::**hospital**()

{*//заполнение* *первоначальных* *данных*

address = "г.Ульяновск, ул.Оренбургская, 27";

name = "ГУЗ Центральная городская клиническая\nбольница г.Ульяновска";

currentId = 7;

Disease\* di1 = *new* Disease;

di1->setNameOfdisease("рак молочной железы");

di1->setSev(0);

di1->setStart(QDate::currentDate());

*//di->setFinish(QDate);*

Disease\* di2 = *new* Disease;

di2->setNameOfdisease("рак щитовидной железы");

di2->setSev(1);

di2->setStart(QDate::currentDate());

Disease\* di3 = *new* Disease;

di3->setNameOfdisease("рак желудочно-кишечного тракта");

di3->setSev(2);

di3->setStart(QDate::currentDate());

Disease\* di4 = *new* Disease;

di4->setNameOfdisease("Пневмония");

di4->setSev(0);

di4->setStart(QDate::currentDate());

*//di->setFinish(QDate);*

Disease\* di5 = *new* Disease;

di5->setNameOfdisease("Ангина");

di5->setSev(1);

di5->setStart(QDate::currentDate());

Disease\* di6 = *new* Disease;

di6->setNameOfdisease("Язва желудка");

di6->setSev(2);

di6->setStart(QDate::currentDate());

Doctor\* doc1=*new* Doctor();

doc1->setName("Иван");

doc1->setSurname("Иванов");

doc1->setPatronymic("Петрович");

doc1->setPosition("заведующий отделением врач-хирург");

doc1->setPolicy("0000111122223333");

Doctor\* doc2=*new* Doctor();

doc2->setName("Петр");

doc2->setSurname("Сидоров");

doc2->setPatronymic("Николаевич");

doc2->setPosition("врач-хирург");

doc2->setPolicy("0000111122224444");

Doctor\* doc3=*new* Doctor();

doc3->setName("Мария");

doc3->setSurname("Рябая");

doc3->setPatronymic("Никитична");

doc3->setPosition("врач-хирург");

doc3->setPolicy("0000111133334444");

Doctor\* doc4=*new* Doctor();

doc4->setName("Евгения");

doc4->setSurname("Иванова");

doc4->setPatronymic("Александровна");

doc4->setPosition("заведующий отделением врач-педиатр");

doc4->setPolicy("0000111122225555");

Doctor\* doc5=*new* Doctor();

doc5->setName("Екатерина");

doc5->setSurname("Майонкова");

doc5->setPatronymic("Алексеевна");

doc5->setPosition("врач-педиатр");

doc5->setPolicy("0000111122226666");

Doctor\* doc6=*new* Doctor();

doc6->setName("Игорь");

doc6->setSurname("Пышков");

doc6->setPatronymic("Викторович");

doc6->setPosition("врач-педиатр");

doc6->setPolicy("0000111133337777");

Patient\* p1=*new* Patient;

p1->setId(1);

p1->setName("Марина");

p1->setSurname("Петрова");

p1->setPatronymic("Александровна");

p1->setPolicy("9999111133334444");

p1->setDoc(*doc1*);

p1->setPdiseas(*di1*);

p1->setPdiseas(*di6*);

p1->setInWard(1);

Patient\* p2=*new* Patient;

p2->setId(2);

p2->setName("Александр");

p2->setSurname("Алексеев");

p2->setPatronymic("Алексеевич");

p2->setPolicy("9999111133335555");

p2->setDoc(*doc2*);

p2->setPdiseas(*di2*);

p2->setInWard(1);

Patient\* p3=*new* Patient;

p3->setId(3);

p3->setName("Алексей");

p3->setSurname("Сафронов");

p3->setPatronymic("Владимирович");

p3->setPolicy("9999111133336666");

p3->setDoc(*doc3*);

p3->setPdiseas(*di3*);

p3->setInWard(1);

Patient\* p4=*new* Patient;

p4->setId(4);

p4->setName("Александра");

p4->setSurname("Петровичева");

p4->setPatronymic("Валентиновна");

p4->setPolicy("9999111133334477");

p4->setDoc(*doc4*);

p4->setPdiseas(*di4*);

p4->setInWard(1);

Patient\* p5=*new* Patient;

p5->setId(5);

p5->setName("Александр");

p5->setSurname("Бородаев");

p5->setPatronymic("Петрович");

p5->setPolicy("9999111133335522");

p5->setDoc(*doc5*);

p5->setPdiseas(*di5*);

p5->setInWard(1);

Patient\* p6=*new* Patient;

p6->setId(6);

p6->setName("Матвей");

p6->setSurname("Салицилов");

p6->setPatronymic("Анатольевич");

p6->setPolicy("9999111133336688");

p6->setDoc(*doc6*);

p6->setPdiseas(*di6*);

p6->setInWard(1);

District\* dis1 = *new* District();

dis1->setNamedistrict("Заволжский район");

dis1->setDpatient(*p1*);

dis1->setDpatient(*p2*);

*//dis1->setDpatient(p3);*

District\* dis2 = *new* District();

dis2->setNamedistrict("Ленинский район");

//dis2-*>setDpatient(p6);*

dis2->setDpatient(*p4*);

dis2->setDpatient(*p5*);

Ward\* war=*new* Ward();

war->setNameward("Хирургическое отделение");

war->setNumOfBed(15);

war->setDoctor(*doc1*);

war->setDoctor(*doc2*);

war->setWpatient(*p1*);

war->setWpatient(*p2*);

Ward\* war2=*new* Ward();

war2->setNameward("Педиатрическое отделение");

war2->setNumOfBed(30);

war2->setDoctor(*doc4*);

war2->setDoctor(*doc5*);

war2->setWpatient(*p4*);

war2->setWpatient(*p5*);

setWard(*war*);

setWard(*war2*);

setDistrict(*dis1*);

setDistrict(*dis2*);

}

int hospital::**getCurrentId**() *const*

{

*return* currentId;

}

void hospital::**setCurrentId**(int newCurrentId)

{

currentId = newCurrentId;

}

*const* QList<District \*> &hospital::**getListOfDistrict**() *const*

{

*return* ListOfDistrict;

}

*const* QList<Ward \*> &hospital::**getListOfWard**() *const*

{

*return* ListOfWard;

}

District \* hospital::**getDistrict**(int n)

{

*return* ListOfDistrict.at(n);

}

void hospital::**getDistrictAndPut**(QString str,Patient\* p)

{

int i(0);

foreach(District\* d, ListOfDistrict){

*if*(d->getNamedistrict()==str)

ListOfDistrict.at(i)->setDpatient(*p*);

i++;

}

}

Ward\* hospital::**getWardByName**(QString strname)

{

int i(0);

foreach(Ward\* w, ListOfWard){

*if*(w->getNameward()==strname){

*return* ListOfWard.at(i);

}

i++;

}

*return* *nullptr*;

}

void hospital::**setDistrict**(District \* &newDistrict)

{

ListOfDistrict.append( newDistrict);

}

int hospital::**getNumOfWard**()

{

*return* ListOfWard.count();

}

int hospital::**getNumOfDistrict**()

{

*return* ListOfDistrict.count();

}

Ward \* hospital::**getWard**(int n)

{

*return* ListOfWard.at(n);

}

void hospital::**setWard**(Ward \* &newWard)

{

ListOfWard.append(newWard);

}

void hospital::**delWard**(int n)

{

ListOfWard.takeAt(n);

}

void hospital::**delDistrict**(int n)

{

ListOfDistrict.takeAt(n);

}

void hospital::**delPatient**(int id)

{

foreach(District\* d,ListOfDistrict){

d->delDpatientId(id);

}

foreach(Ward\* w,ListOfWard){

w->delPatientId(id);

}

}

void hospital::**delPatientfromWard**(int id)

{

foreach(Ward\* w,ListOfWard){

w->delPatientId(id);

}

}

bool hospital::**DoctorhasPatient**(Doctor\* d){

foreach(Ward\* w,ListOfWard){

foreach(Patient\* p,w->getWpatients()){

*if*(p->getDoc()->getPolicy()==d->getPolicy())

*return* 1;

}

}

*return* 0;

}

QList<Patient\*>hospital::**DoctorhasPatients**(Doctor\* d){

QList<Patient\*> list;

foreach(Ward\* w,ListOfWard){

foreach(Patient\* p,w->getWpatients()){

*if*(p->getDoc()->getPolicy()==d->getPolicy())

list.append(p);

}

}

*return* list;

}

void hospital::**delDoctor**(QString pol)

{

foreach(Ward\* w,ListOfWard){

w->delDpoctorPol(pol);

}

}

Patient\* hospital::**getPatientPolicy**(QString pol){

foreach(District\* d,ListOfDistrict){

foreach(Patient\* p,d->getDpatients())

*if*(p->getPolicy()==pol)

*return* p;

}

*return* *nullptr*;

}

Doctor\* hospital::**getDoctorPolicy**(QString pol){

foreach(Ward\* w,ListOfWard){

foreach(Doctor\* d,w->getDoctors())

*if*(d->getPolicy()==pol)

*return* d;

}

*return* *nullptr*;

}

*const* QString &hospital::**getName**() *const*

{

*return* name;

}

void hospital::**setName**(*const* QString &newName)

{

name = newName;

}

*const* QString &hospital::**getAddress**() *const*

{

*return* address;

}

void hospital::**setAddress**(*const* QString &newAddress)

{

address = newAddress;

}

hospital::~**hospital**()

{

qDeleteAll(ListOfDistrict);

qDeleteAll(ListOfWard);

}

**Human.h**

#ifndef HUMAN\_H

#define HUMAN\_H

#include "QString"

*class* **Human**

{

*public*:

**Human**();

*//сеттеры* *и* *геттеры* *для* *хранимых* *данных*

void **setName**(*const* QString &newName);

void **setSurname**(*const* QString &newSurname);

void **setPatronymic**(*const* QString &newPatronymic);

*const* QString &**getName**() *const*;

*const* QString &**getSurname**() *const*;

*const* QString &**getPatronymic**() *const*;

void **setPolicy**(*const* QString &newPolicy);

*const* QString &**getPolicy**() *const*;

*protected*:

QString Name; *//имя*

QString Surname;*//фамилия*

QString Patronymic;*//отчество*

QString policy; *//номер* *полиса* *медицинского* *страхования*

};

#endif *//* *HUMAN\_H*

**Human.cpp**

#include "human.h"

Human::**Human**()

{

}

void Human::**setName**(*const* QString &newName)

{

Name = newName;

}

void Human::**setSurname**(*const* QString &newSurname)

{

Surname = newSurname;

}

void Human::**setPatronymic**(*const* QString &newPatronymic)

{

Patronymic = newPatronymic;

}

void Human::**setPolicy**(*const* QString &newPolicy)

{

policy = newPolicy;

}

*const* QString &Human::**getName**() *const*

{

*return* Name;

}

*const* QString &Human::**getSurname**() *const*

{

*return* Surname;

}

*const* QString &Human::**getPatronymic**() *const*

{

*return* Patronymic;

}

*const* QString &Human::**getPolicy**() *const*

{

*return* policy;

}

**MainWindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include"hospital.h"

#include "dialog\_ward.h"

#include "dialog\_patient.h"

#include "dialog\_district.h"

#include "dialog\_doctor.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

signals:

void **sendData**(hospital\* H);*//сигнал* *который* *отправляет* *данные*

*private* slots:

void **on\_ward\_clicked**();*//кнопка* *для* *открытия* *окна* *с* *отделениями* *и* *передачи* *в* *новый* *класс* *данных*

void **on\_patient\_clicked**();*//кнопка* *для* *открытия* *окна* *с* *пациентами* *и* *передачи* *в* *новый* *класс* *данных*

void **on\_district\_clicked**();*//кнопка* *для* *открытия* *окна* *с* *районами* *и* *передачи* *в* *новый* *класс* *данных*

void **on\_doctor\_clicked**();*//кнопка* *для* *открытия* *окна* *с* *врачами* *и* *передачи* *в* *новый* *класс* *данных*

*private*:

Ui::MainWindow \*ui;

Dialog\_ward \*dw;*//диалоговое* *окно* *меню* *для* *отделений*

Dialog\_patient \*dp;*//диалоговое* *окно* *меню* *для* *пациентов*

Dialog\_district \*dd;*//диалоговое* *окно* *меню* *для* *районов*

Dialog\_doctor \*ddoc;*//диалоговое* *окно* *меню* *для* *врачей*

hospital\* Hospital = hospital::gethospital();*//объект* *класса* *больница*

};

#endif *//* *MAINWINDOW\_H*

**MainWindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "dialog\_ward.h"

#include "dialog\_patient.h"

#include "dialog\_district.h"

#include "dialog\_doctor.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

ui->name->setText(Hospital->getName());

ui->address->setText(Hospital->getAddress());

dw = *new* Dialog\_ward();

dp = *new* Dialog\_patient();

dd = *new* Dialog\_district();

ddoc = *new* Dialog\_doctor();

*//связываем* *сигнал* *и* *слот* *новой* *формы,* *который* *будет* *принимать* *данные*

connect(*this*, SIGNAL(sendData(hospital\*)), dw, SLOT(recieveData(hospital\*)));

connect(*this*, SIGNAL(sendData(hospital\*)), dp, SLOT(recieveData(hospital\*)));

connect(*this*, SIGNAL(sendData(hospital\*)), dd, SLOT(recieveData(hospital\*)));

connect(*this*, SIGNAL(sendData(hospital\*)), ddoc, SLOT(recieveData(hospital\*)));

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

void MainWindow::**on\_ward\_clicked**()

{

dw->show();

emit sendData(*Hospital*);*//сигнал* *для* *передачи* *данных* *на* *другую* *форму*

}

void MainWindow::**on\_patient\_clicked**()

{

dp->show();

emit sendData(*Hospital*);*//сигнал* *для* *передачи* *данных* *на* *другую* *форму*

}

void MainWindow::**on\_district\_clicked**()

{

dd->show();

emit sendData(*Hospital*);*//сигнал* *для* *передачи* *данных* *на* *другую* *форму*

}

void MainWindow::**on\_doctor\_clicked**()

{

ddoc->show();

emit sendData(*Hospital*);*//сигнал* *для* *передачи* *данных* *на* *другую* *форму*

}

**Patient.h**

#ifndef PATIENT\_H

#define PATIENT\_H

#include"QString"

#include"QList"

#include"human.h"

#include"doctor.h"

#include"disease.h"

*class* **Patient**: *public* Human

{

*public*:

**Patient**();

~**Patient**();

*//сеттеры* *и* *геттеры* *для* *хранимых* *данных*

Doctor \***getDoc**() *const*;

void **setDoc**(Doctor \*newDoc);

*const* QList<Disease \*> &**getPdiseases**() *const*;

void **setPdiseas**(Disease \*&newPdisease);

int **getId**() *const*;

void **setId**(int newId);

bool **getInWard**() *const*;

void **setInWard**(bool newInWard);

*//получение* *числа* *заболеваний*

int **getNumOfDisease**();

*//получение* *болезни* *по* *порядковому* *номеру* *в* *списке*

Disease \***getPdiseas**(int n);

*//проверка* *есть* *ли* *болезни* *в* *списке*

bool **hasDiseases**();

*private*:

int id; *//идентификатор*

bool inWard=0; *//находится* *в* *отделении* *или* *нет*

Doctor\* doc; *//прикрепленный* *врач*

QList<Disease\*> pdiseases;*//список* *болезней*

*//список* *болезней* *представляет* *собой* *импровизированную* *амбулаторную* *карту*

*//в* *ней* *хранится* *информация* *о* *истории* *болезней*

};

#endif *//* *PATIENT\_H*

**Patient.cpp**

#include "patient.h"

Patient::**Patient**()

{

}

Patient::~**Patient**()

{

qDeleteAll(pdiseases);

*delete* doc;

}

bool Patient::**hasDiseases**(){

*if*(pdiseases.empty())

*return* *false*;

*else*

*return* *true*;

}

Disease\* Patient::**getPdiseas**(int n)

{

*return* pdiseases.at(n);

}

void Patient::**setPdiseas**(Disease\* &newPdisease)

{

pdiseases.append(newPdisease);

}

int Patient::**getNumOfDisease**()

{

*return* pdiseases.count();

}

int Patient::**getId**() *const*

{

*return* id;

}

void Patient::**setId**(int newId)

{

id = newId;

}

bool Patient::**getInWard**() *const*

{

*return* inWard;

}

void Patient::**setInWard**(bool newInWard)

{

inWard = newInWard;

}

*const* QList<Disease \*> &Patient::**getPdiseases**() *const*

{

*return* pdiseases;

}

Doctor \*Patient::**getDoc**() *const*

{

*return* doc;

}

void Patient::**setDoc**(Doctor \*newDoc)

{

doc = newDoc;

}

**Ward.h**

#ifndef WARD\_H

#define WARD\_H

#include "doctor.h"

#include "patient.h"

#include "QString"

#include "QList"

*class* **Ward**

{

*public*:

**Ward**();

~**Ward**();

*//сеттеры* *и* *геттеры* *для* *хранимых* *данных*

*const* QString &**getNameward**() *const*;

void **setNameward**(*const* QString &newNameward);

int **getNumOfBed**() *const*;

void **setNumOfBed**(int newNumOfBed);

*const* QList<QString> &**getDiseases**() *const*;

void **setDiseases**(*const* QList<QString> &newDiseases);

Patient \***getWpatient**(int n);

void **setWpatient**(Patient \*&newWpatient);

Doctor \***getDoctor**(int n) *const*;

void **setDoctor**(Doctor \*&newDoctor);

*const* QList<Patient \*> &**getWpatients**() *const*;

*const* QList<Doctor \*> &**getDoctors**() *const*;

*//получение* *количества* *пациентов*

int **getNumOfPatient**();

*//получение* *количества* *врачей*

int **getNumOfDoctors**();

*//проверка* *есть* *ли* *пациенты* *в* *списке*

bool **hasPatients**();

*//проверка* *есть* *ли* *врачи* *в* *списке*

bool **hasDoctors**();

*//удаление* *пациента* *по* *Id*

void **delPatientId**(int id);

*//удаление* *врача* *по* *номеру* *полиса*

void **delDpoctorPol**(QString pol);

*private*:

QString nameward; *//название* *отделения*

int NumOfBed; *//количество* *коек*

QList <QString> diseases; *//список* *болезней* *с* *которыми* *кладут* *в* *отделение*

QList<Patient\*> wpatients; *//пациенты*

QList<Doctor\*> doctors; *//доктора*

};

#endif *//* *WARD\_H*

**Ward.cpp**

#include "ward.h"

Ward::**Ward**()

{

qDeleteAll(wpatients);

qDeleteAll(doctors);

}

Ward::~**Ward**()

{

qDeleteAll(wpatients);

qDeleteAll(doctors);

}

*const* QString &Ward::**getNameward**() *const*

{

*return* nameward;

}

void Ward::**setNameward**(*const* QString &newNameward)

{

nameward = newNameward;

}

int Ward::**getNumOfBed**() *const*

{

*return* NumOfBed;

}

void Ward::**setNumOfBed**(int newNumOfBed)

{

NumOfBed = newNumOfBed;

}

*const* QList<QString> &Ward::**getDiseases**() *const*

{

*return* diseases;

}

void Ward::**setDiseases**(*const* QList<QString> &newDiseases)

{

diseases = newDiseases;

}

Patient \* Ward::**getWpatient**(int n)

{

*return* wpatients.at(n);

}

void Ward::**setWpatient**(Patient \* &newWpatient)

{

wpatients.append(newWpatient);

}

Doctor \* Ward::**getDoctor**(int n) *const*

{

*return* doctors.at(n);

}

void Ward::**setDoctor**(Doctor \* &newDoctor)

{

doctors.append(newDoctor);

}

int Ward::**getNumOfPatient**()

{

*return* wpatients.count();

}

int Ward::**getNumOfDoctors**()

{

*return* doctors.count();

}

*//проверка* *на* *непустой* *список*

bool Ward::**hasPatients**()

{

*if*(wpatients.empty())

*return* *false*;

*else*

*return* *true*;

}

*//проверка* *на* *непустой* *список*

bool Ward::**hasDoctors**()

{

*if*(doctors.empty())

*return* *false*;

*else*

*return* *true*;

}

void Ward::**delPatientId**(int id)

{

int i(0);

foreach(Patient\* p, wpatients){

*if*(p->getId()==id){

wpatients.takeAt(i);

}

i++;

}

}

void Ward::**delDpoctorPol**(QString pol){

int i(0);

foreach(Doctor\* d, doctors){

*if*(d->getPolicy()==pol){

doctors.takeAt(i);

}

i++;

}

}

*const* QList<Doctor \*> &Ward::**getDoctors**() *const*

{

*return* doctors;

}

*const* QList<Patient \*> &Ward::**getWpatients**() *const*

{

*return* wpatients;

}